Redes de Computadores

Prof. Daniel Ludovico Guidoni
guidoni@ufop.edu.br

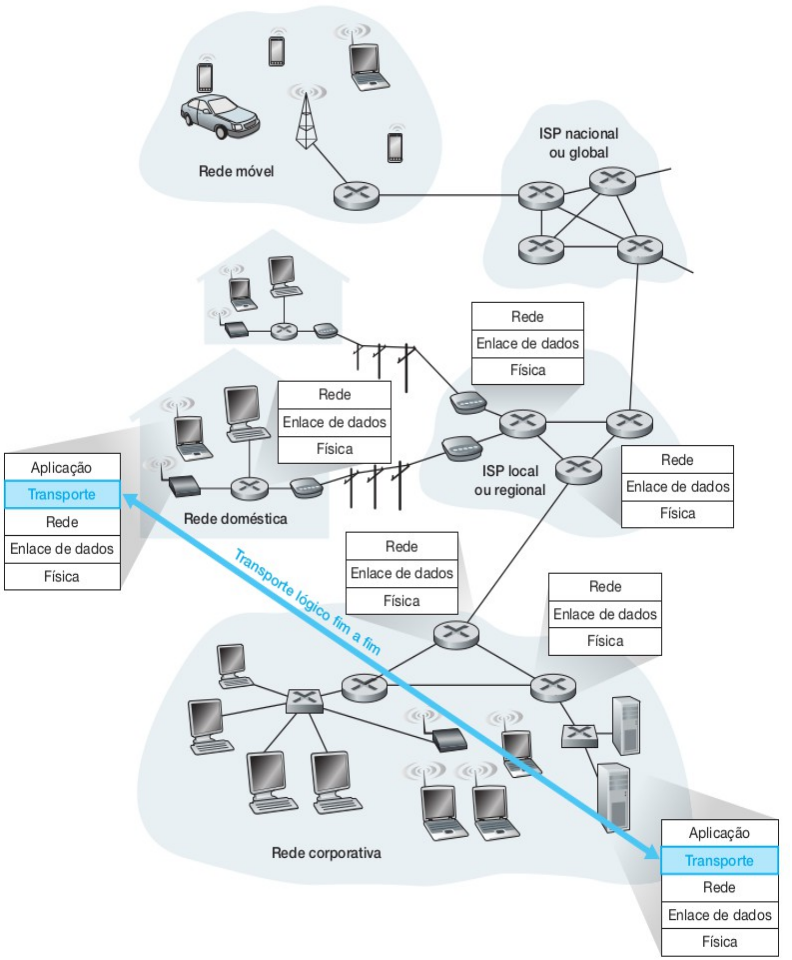
Redes de Computadores

Capítulo 3: Camada de transporte

Aula 01: Introdução e serviços de camada de transporte

Serviços e protocolos de transporte

- Fornecem comunicação lógica entre processos de aplicação executando em diferentes hospedeiros
- Os protocolos de transporte são executados nos sistemas finais:
  - Lado transmissor: quebra as mensagens da aplicação em segmentos, repassa-os para a camada de rede
  - Lado receptor: remonta as mensagens a partir dos segmentos, repassa-as para a camada de aplicação
- Existe mais de um protocolo de transporte disponível para as aplicações
  - Internet: TCP e UDP

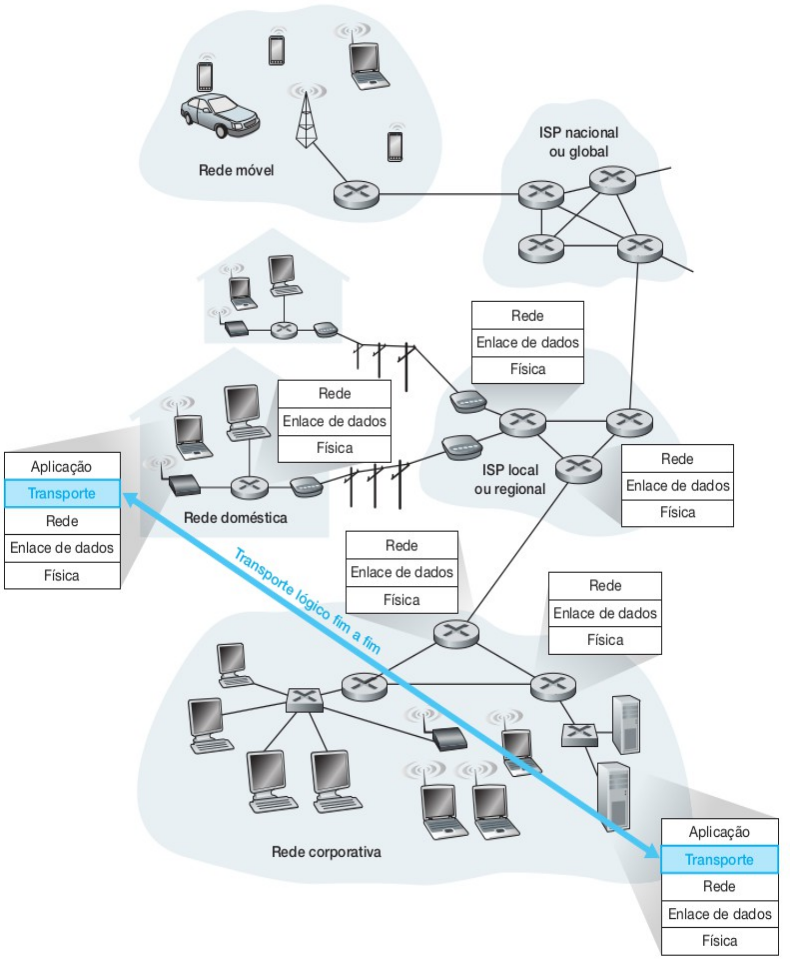


Camadas de transporte *versus* rede

- camada de rede: comunicação lógica entre hospedeiros
- camada de transporte: comunicação lógica entre os processos
  - depende e estende serviços da camada de rede

Protocolos da camada de transporte da Internet

- TCP: Transmission Control Protocol
  - Entrega confiável e ordenada
  - Controle de congestionamento
  - Controle de fluxo
  - Estabelecimento de conexão
- UDP: User Datagram Protocol
  - Entrega não confiável e não ordenada
  - Extensão direta do “melhor esforço” do IP

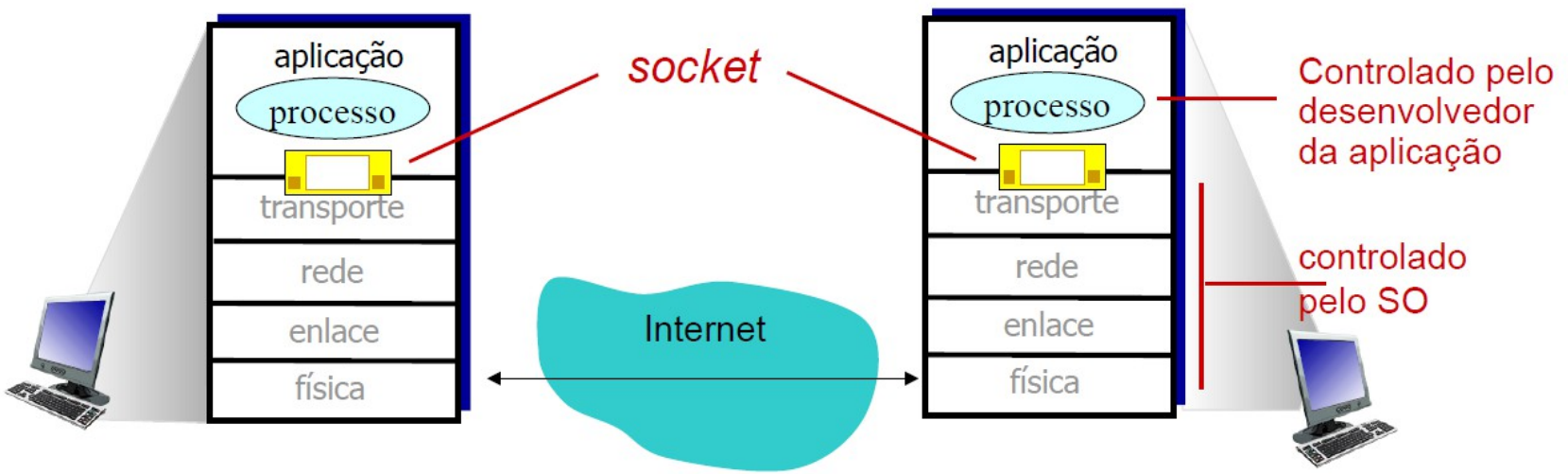


Protocolos da camada de transporte da Internet

- Serviços NÃO disponíveis
  - Garantias de atraso máximo
  - Garantias de largura de banda mínimas

Socket

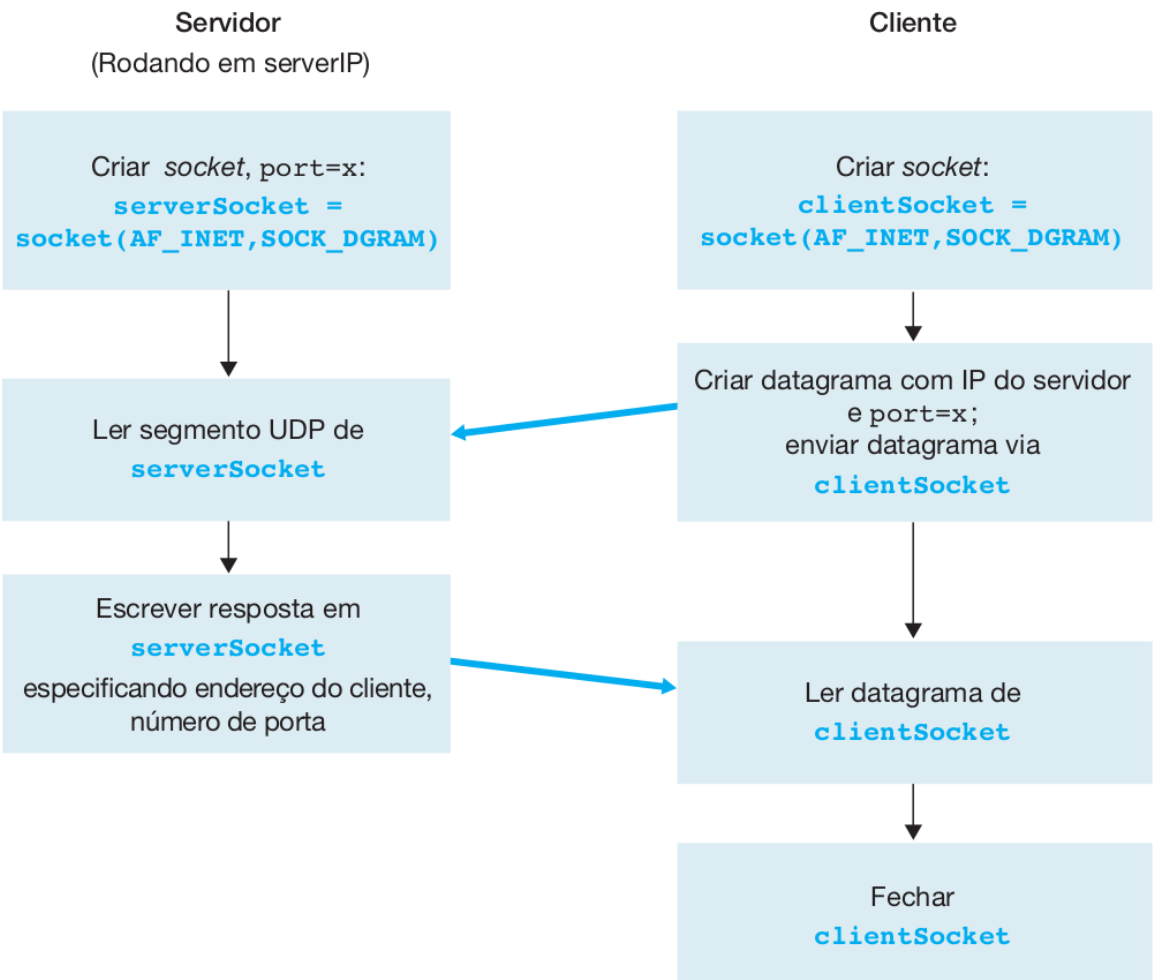
- Criando aplicações em rede



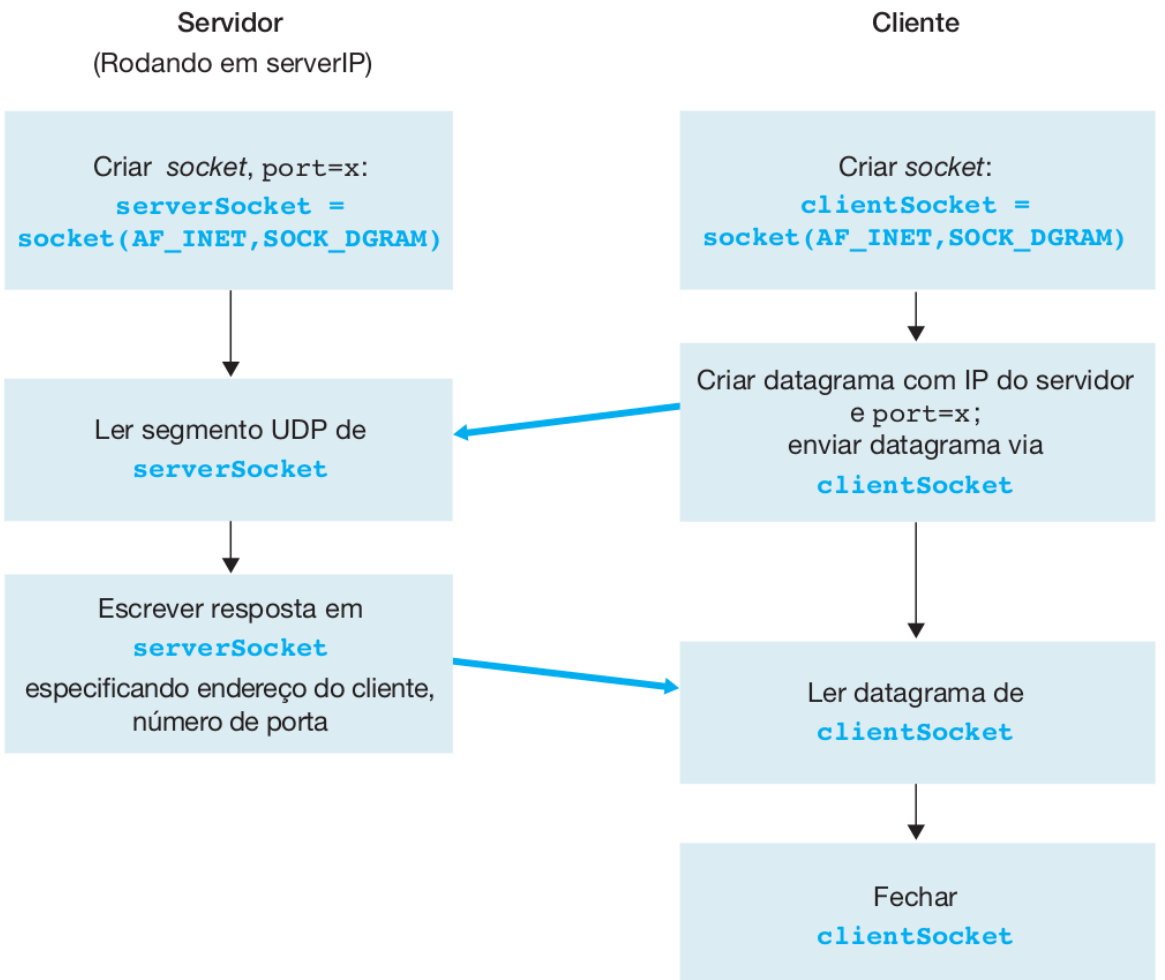
Sockets

- Dois tipos de sockets para dois serviços de transporte
  - UDP:datagrama não confiável
  - TCP:confiável, orientado a fluxos de bytes

Socket UDP

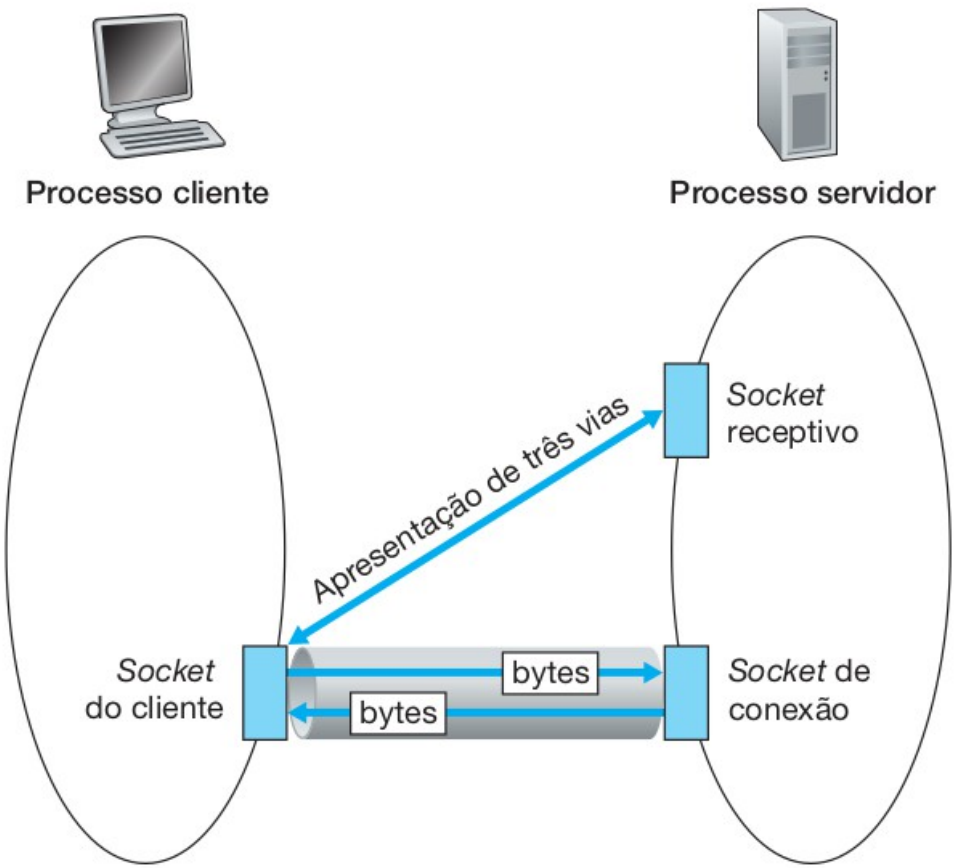

Servidor
(Rodando em serverIP)
Cliente
Criar *socket*, port=x:
serverSocket =
socket(AF_INET,SOCK_DGRAM)
Criar *socket*:
clientSocket =
socket(AF_INET,SOCK_DGRAM)
Criar datagrama com IP do servidor
e port=x;
enviar datagrama via
clientSocket
Ler segmento UDP de
serverSocket
Escrever resposta em
serverSocket
especificando endereço do cliente,
número de porta
Ler datagrama de
clientSocket
Fechar
clientSocket

Socket UDP

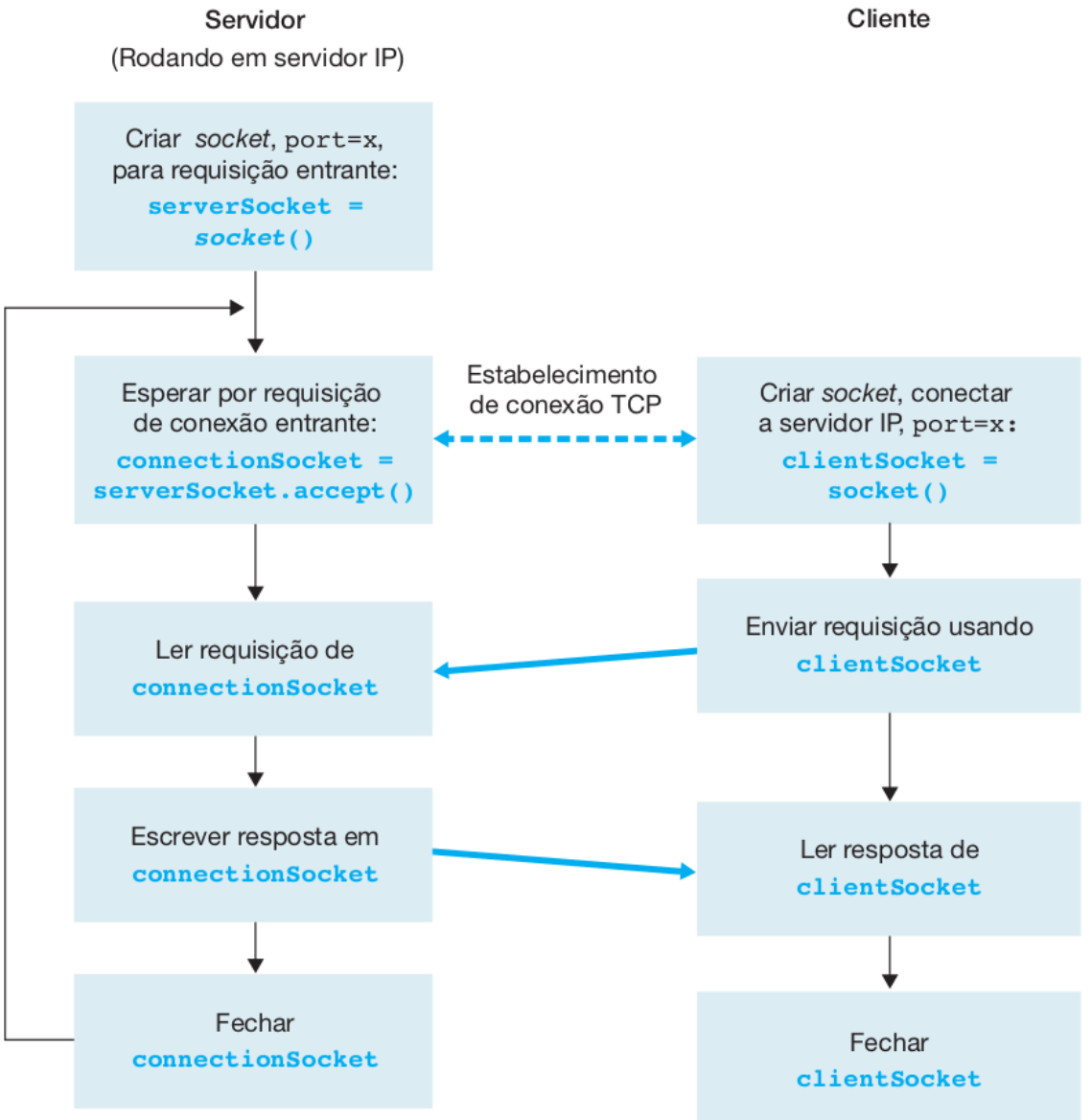


Dificuldade: eventos em Computadores diferentes

Socket TCP


Processo cliente
Processo servidor
Socket receptivo
Apresentação de três vias
Socket do cliente
bytes
bytes
Socket de conexão

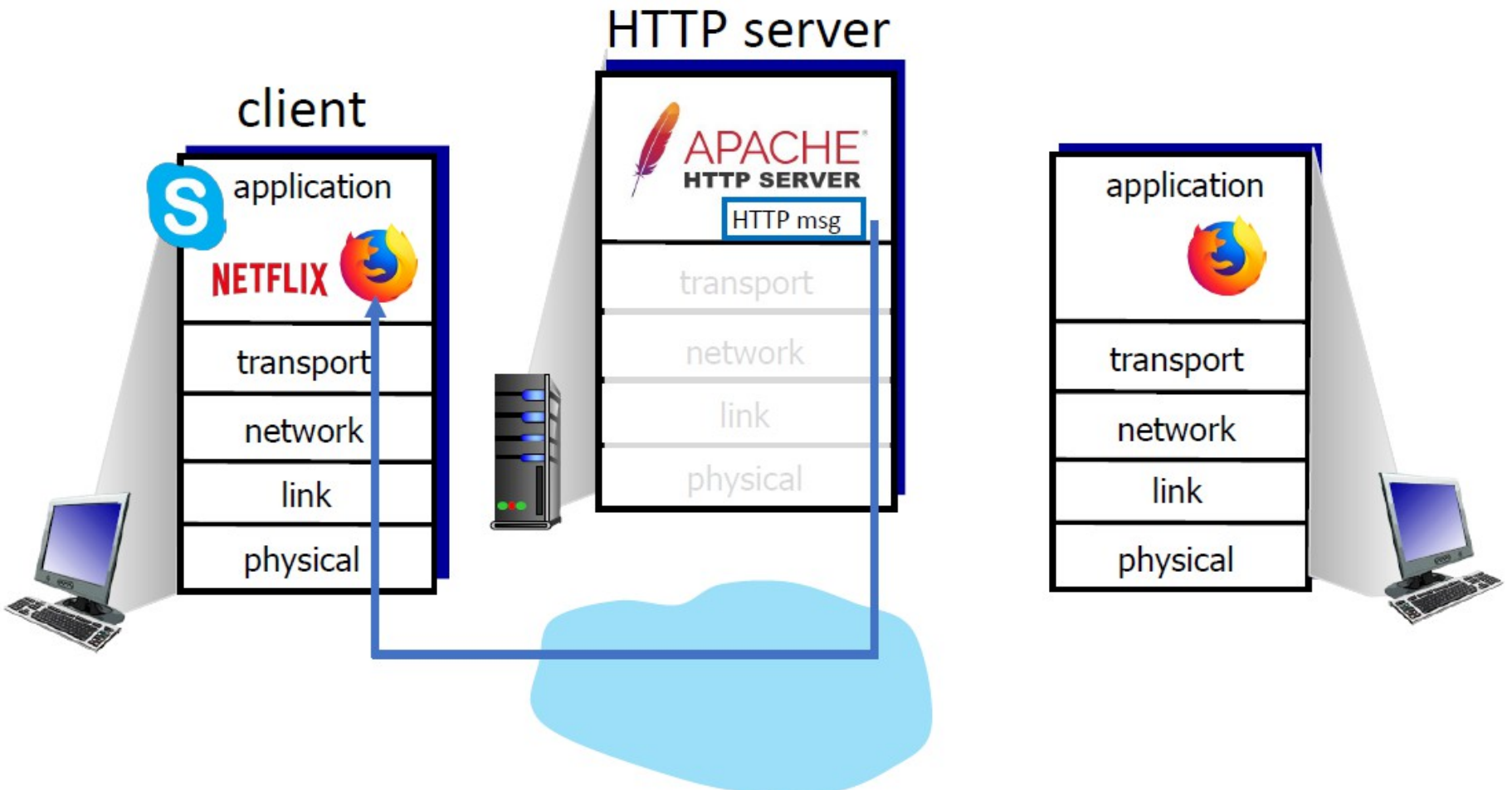
Socket TCP

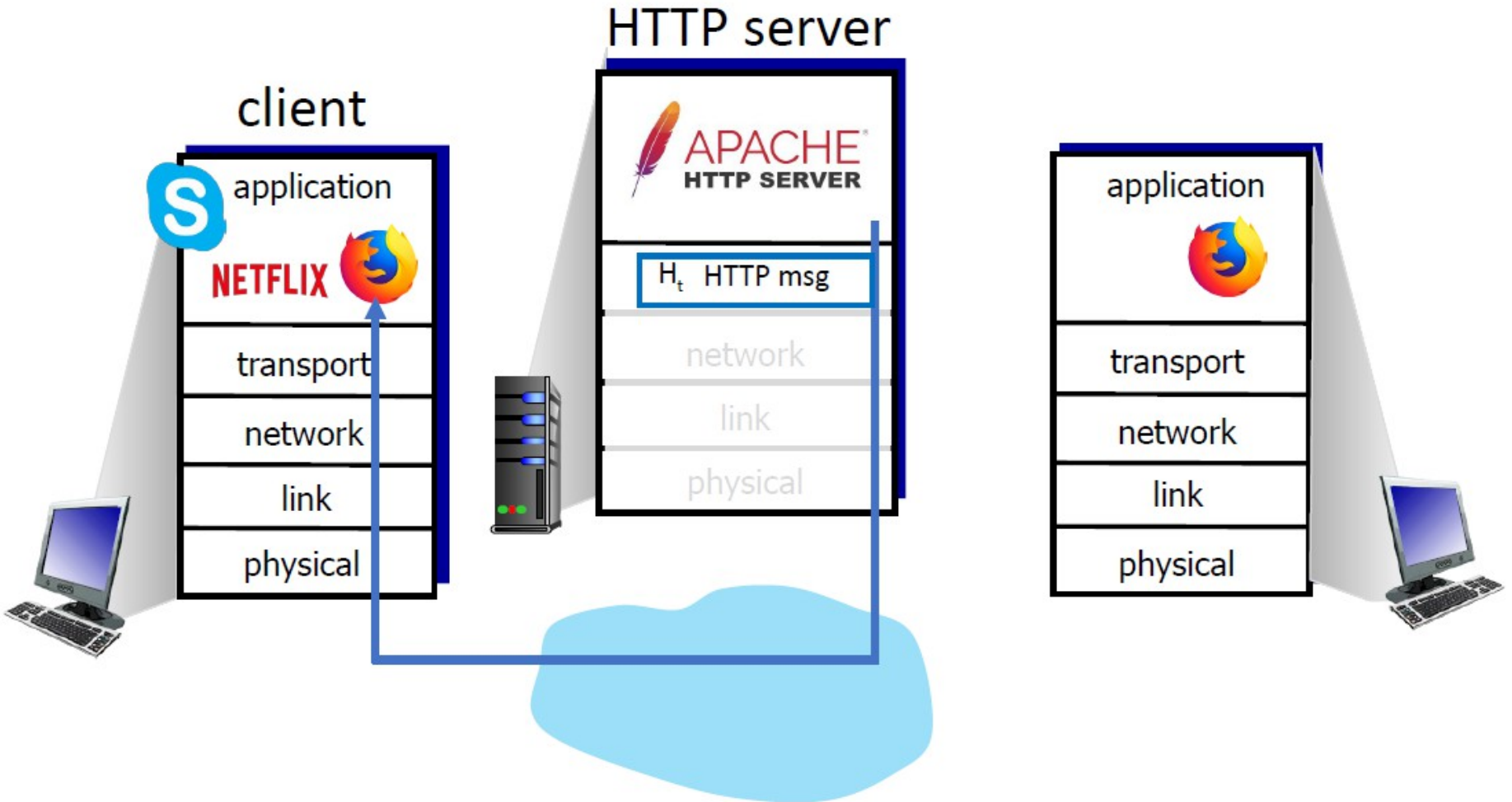

Servidor
(Rodando em servidor IP)
Cliente
Criar socket, port=x, para requisição entrante:
serverSocket = socket()
Esperar por requisição de conexão entrante:
connectionSocket = serverSocket.accept()
Estabelecimento de conexão TCP
Criar socket, conectar a servidor IP, port=x:
clientSocket = socket()
Ler requisição de
connectionSocket
Enviar requisição usando
clientSocket
Escrever resposta em
connectionSocket
Ler resposta de
clientSocket
Fechar
connectionSocket
Fechar
clientSocket

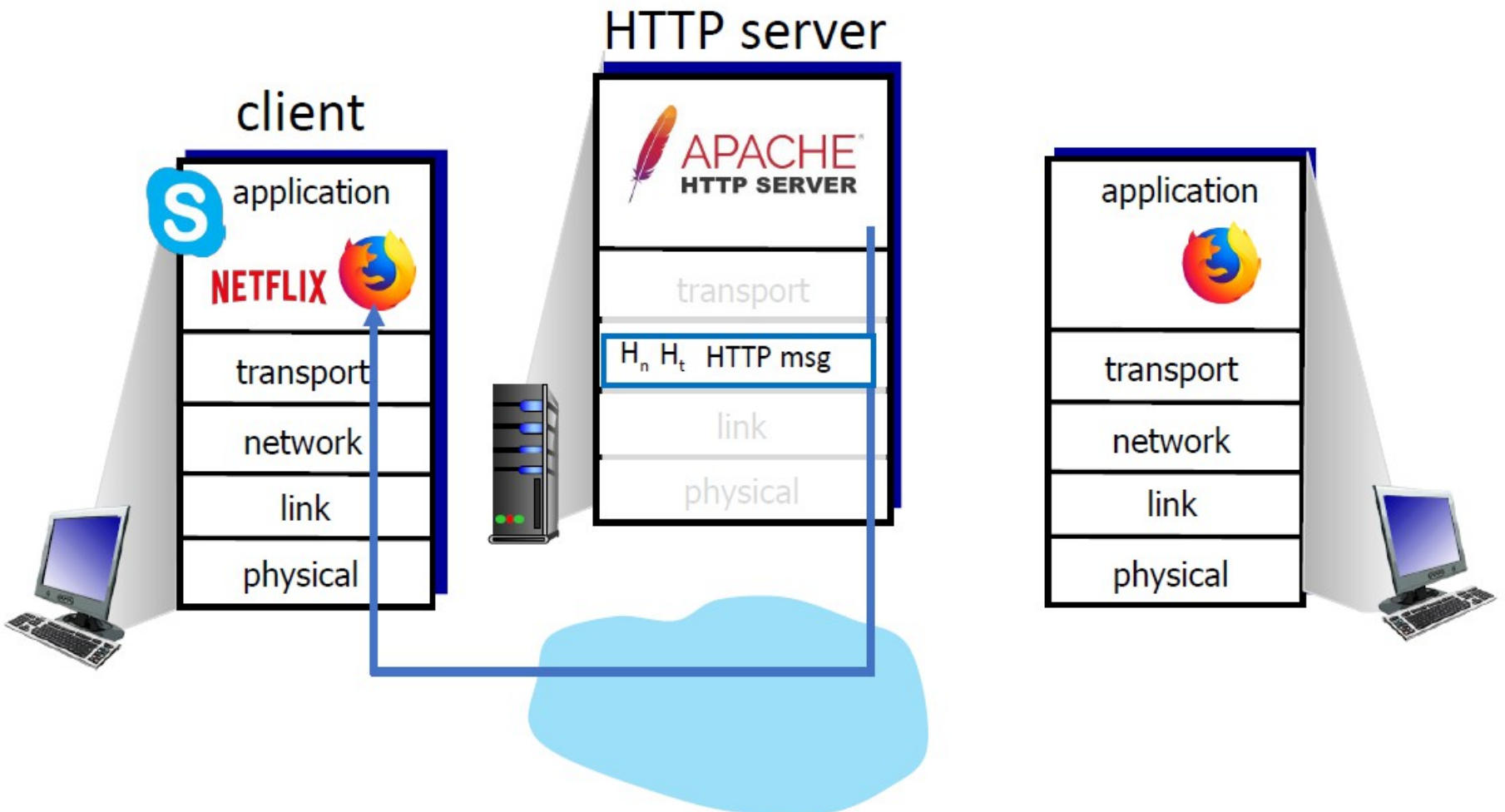
Redes de Computadores

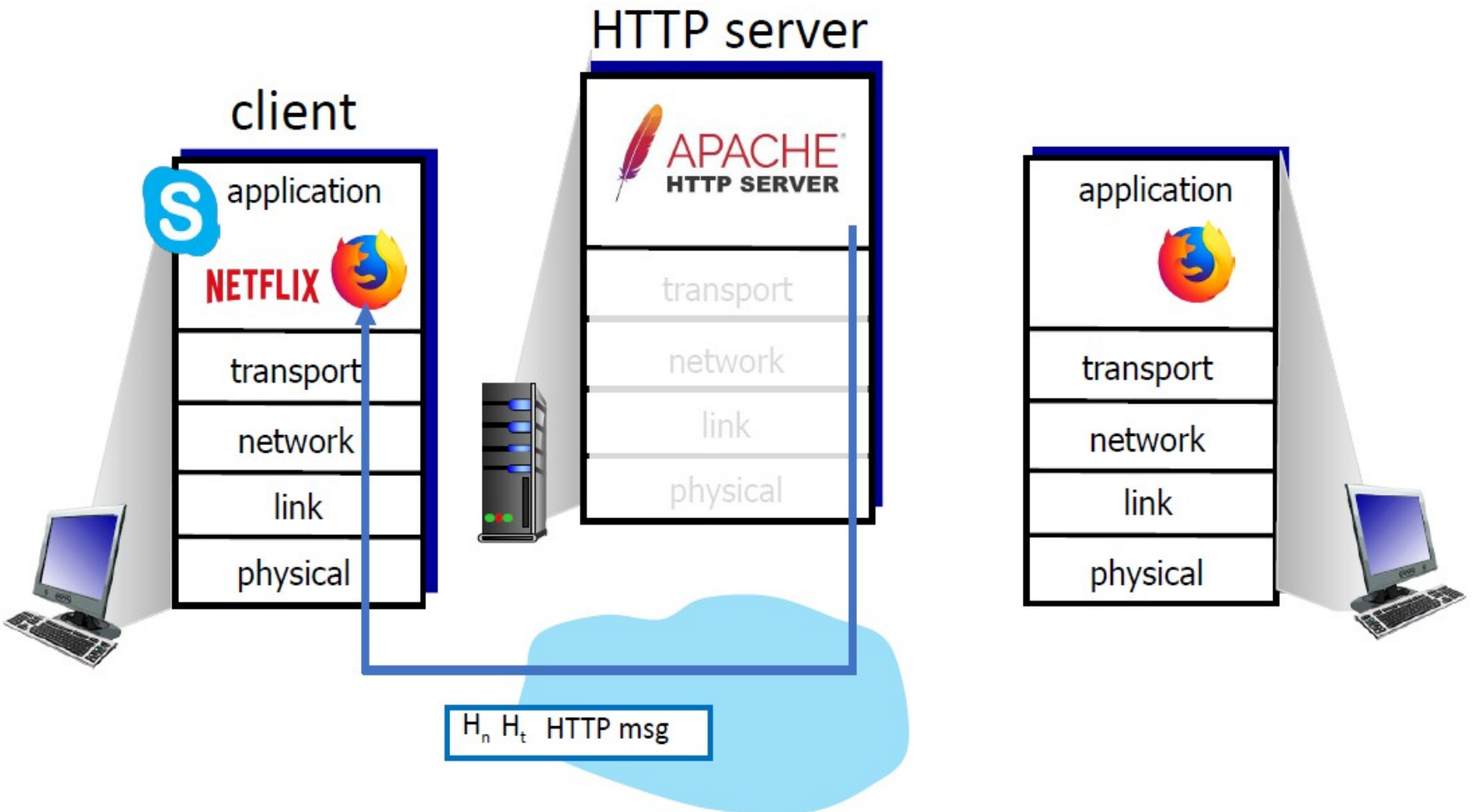
Capítulo 3: Camada de transporte

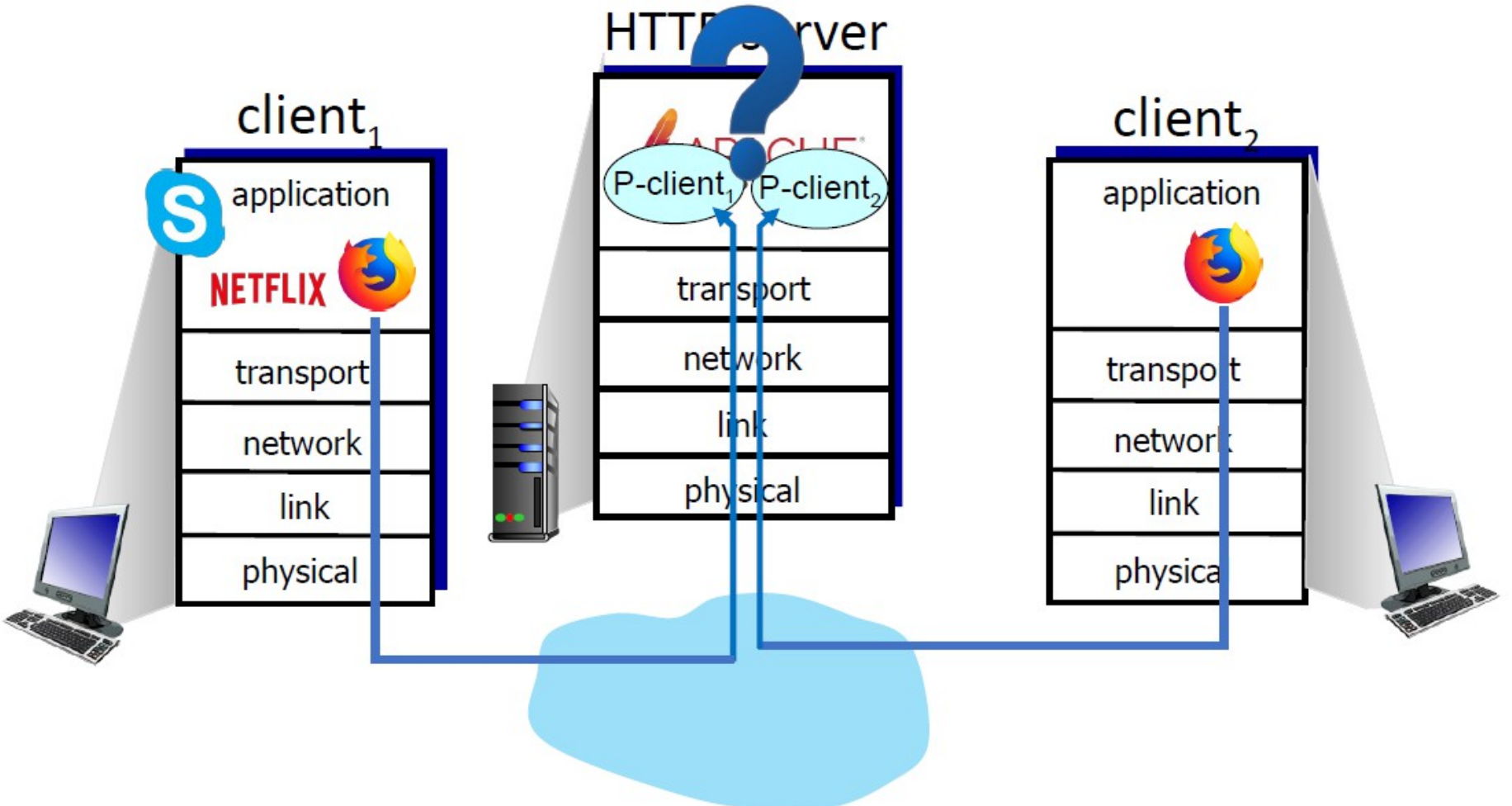
Multiplexação e demultiplexação


HTTP server
client
application
NETFLIX
transport
network
link
physical
APACHE
HTTP SERVER
HTTP msg
transport
network
link
physical
application
transport
network
link
physical


HTTP server
client
application
NETFLIX
transport
network
link
physical
APACHE
HTTP SERVER
H_t HTTP msg
network
link
physical
application
transport
network
link
physical


HTTP server
client
application
NETFLIX
transport
network
link
physical
APACHE
HTTP SERVER
transport
H_n H_t HTTP msg
link
physical
application
transport
network
link
physical

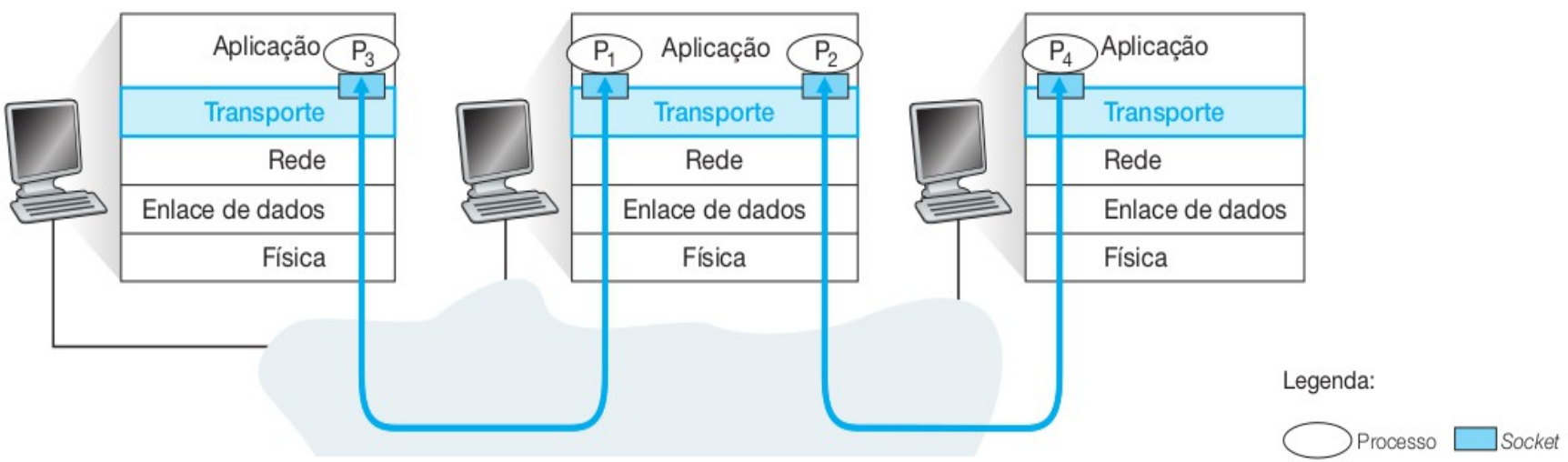

client
application
NETFLIX
transport
network
link
physical
HTTP server
APACHE
HTTP SERVER
transport
network
link
physical
application
transport
network
link
physical
H_n H_t HTTP msg


client1
application
NETFLIX
transport
network
link
physical
HTTP server
P-client1
P-client2
transport
network
link
physical
client2
application
transport
network
link
physical

Multiplexação e Demultiplexação

Multiplexação no transmissor:
reúne dados de muitos sockets, adiciona o cabeçalho de transporte (usado posteriormente para a demultiplexação)

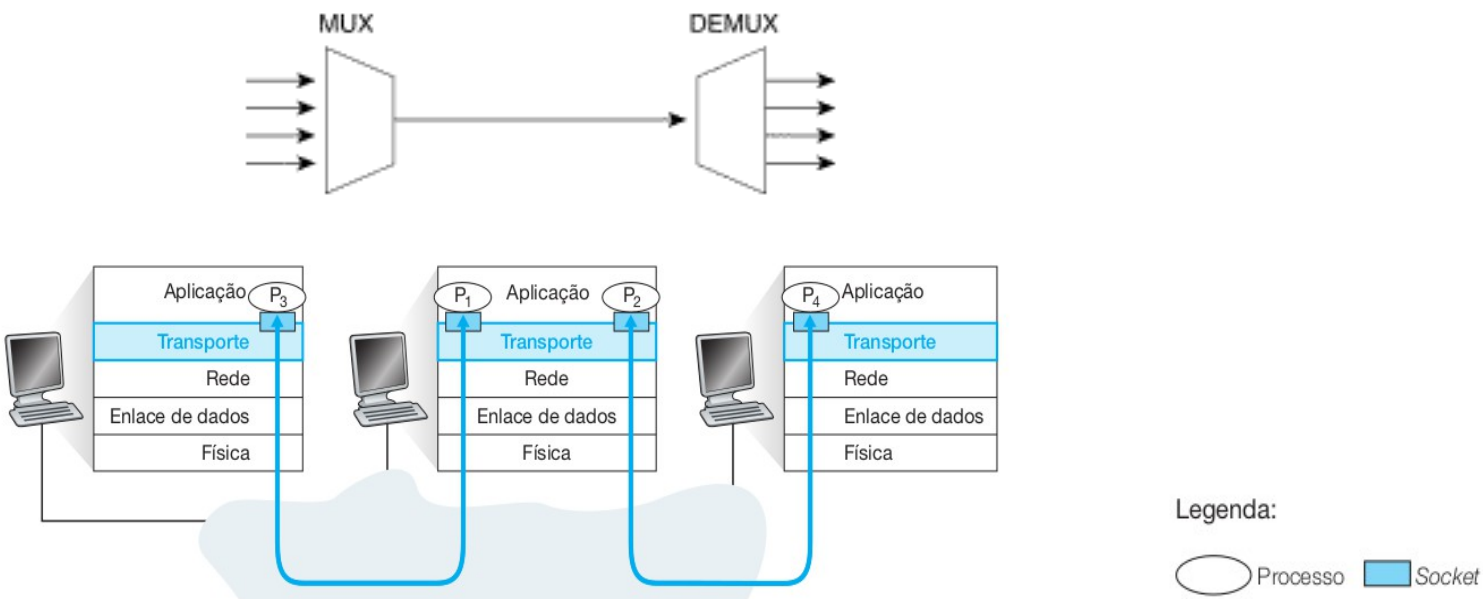
Demultiplexação no receptor:
Usa info do cabeçalho para entregar os segmentos recebidos aos sockets corretos



Multiplexação e Demultiplexação

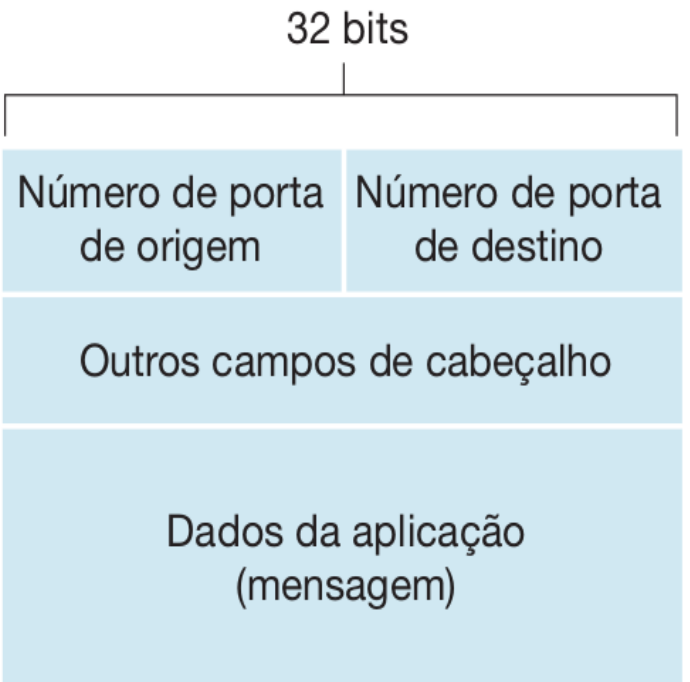
Multiplexação no transmissor:
reúne dados de muitos sockets, adiciona o cabeçalho de transporte (usado posteriormente para a demultiplexação)

Demultiplexação no receptor:
Usa info do cabeçalho para entregar os segmentos recebidos aos sockets corretos



Como funciona a demultiplexação

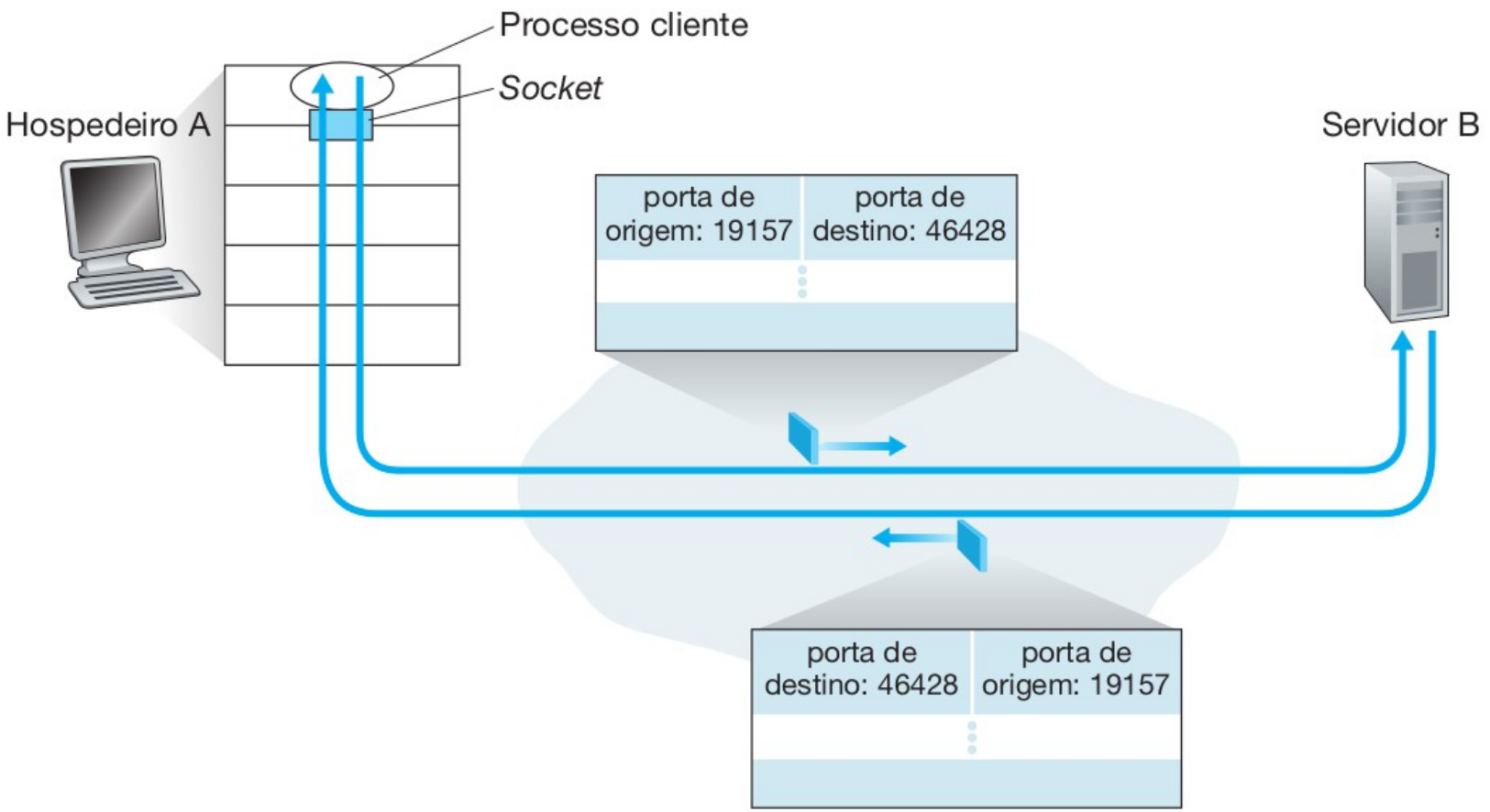
- computador recebe os datagramas IP
  - cada datagrama possui os endereços IP da origem e do destino
  - cada datagrama transporta um segmento da camada de transporte
  - cada segmento possui números das portas origem e destino
- O hospedeiro usa os endereços IP e os números das portas para direcionar o segmento ao socket apropriado



Demultiplexação não orientada a conexões

- Remetente:
  - Socket criado possui número da posta local ao host
  - Ao criar um datagrama para enviar ao destinatário, deve especificar:
    - Endereço IP do destino
    - Número de porta do destino
- Destinatário
  - Quando um host recebe um datagrama
    - Verifica o número da porta de destino contida no pacote
    - Encaminha o datagrama para o socket/processo com aquele número de porta

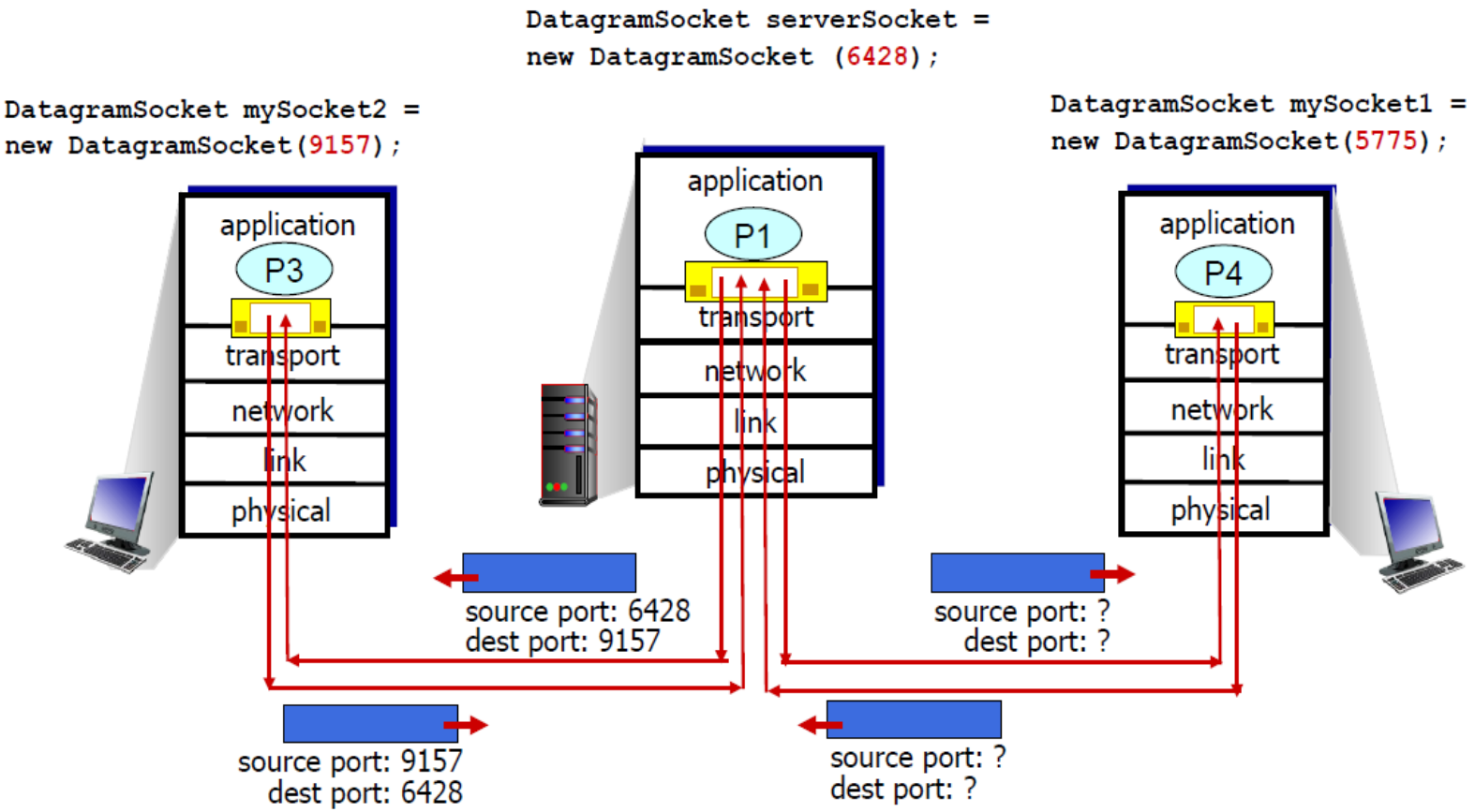
Demultiplexação não orientada a conexões



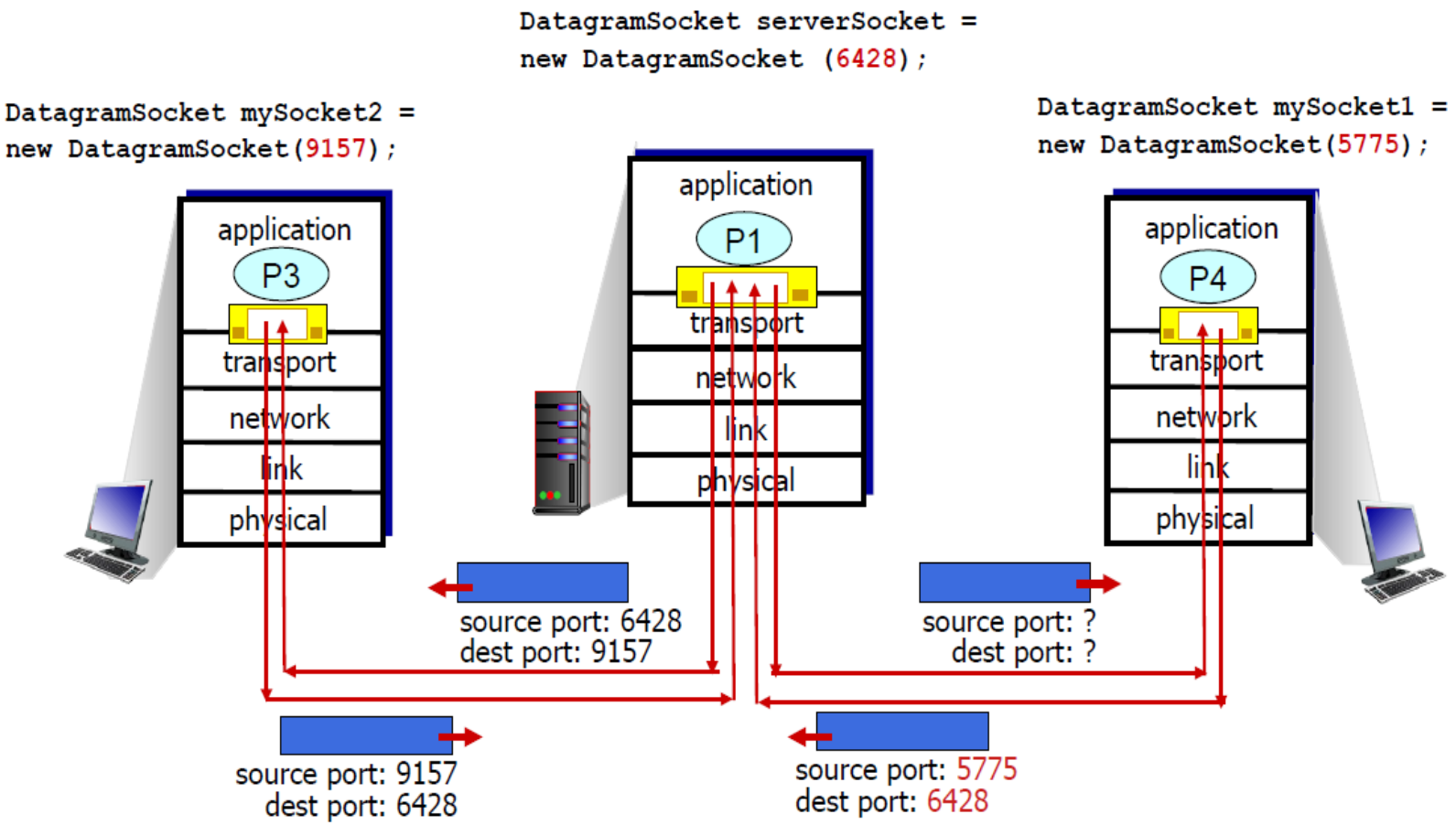
Demultiplexação não orientada a conexões

- Datagramas IP com mesmo número de porta destino, mas diferentes endereços IP origem e/ou números de porta origem podem ser encaminhados para o mesmo socket no destino

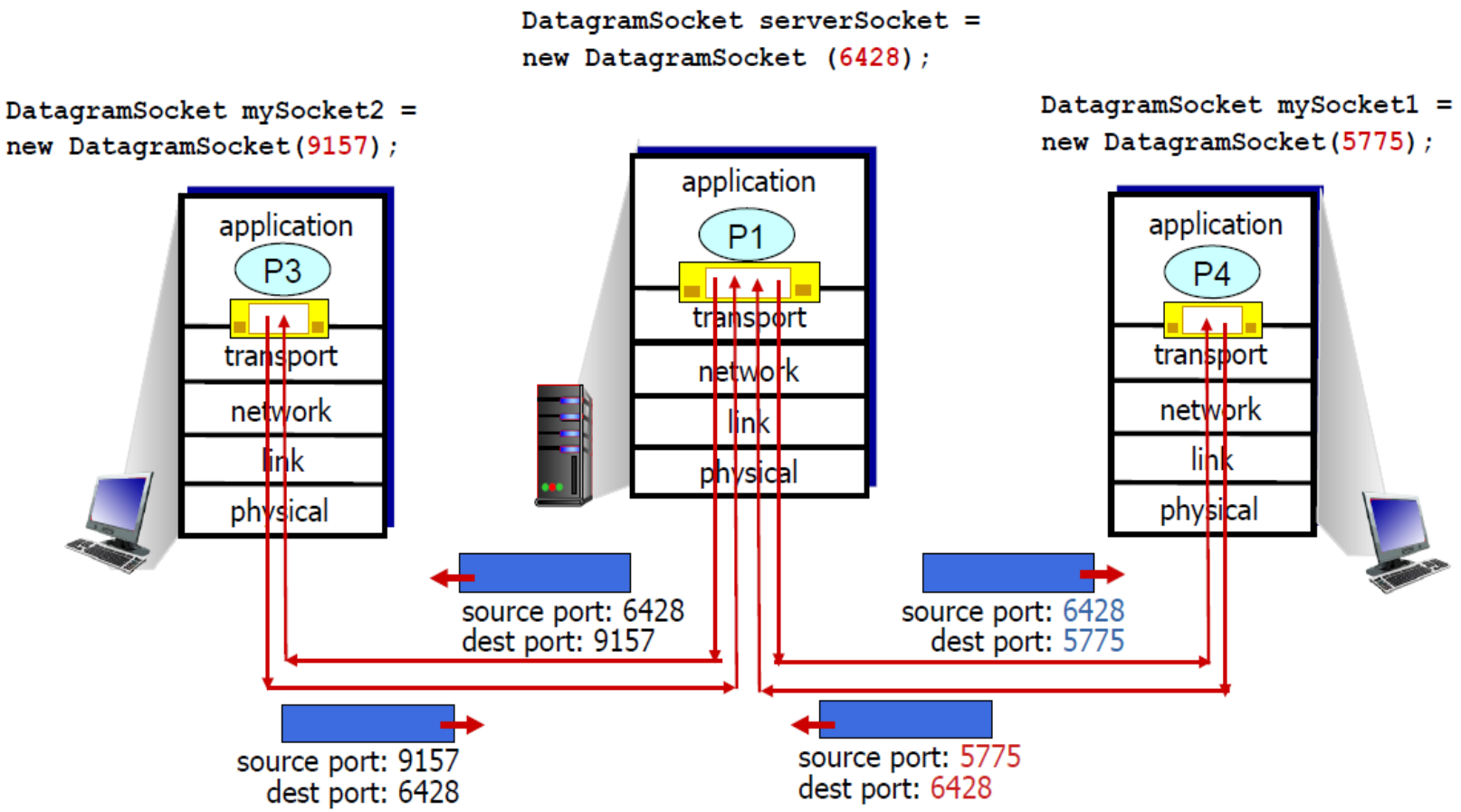
Exemplo


DatagramSocket serverSocket =
new DatagramSocket (6428);
DatagramSocket mySocket2 =
new DatagramSocket(9157);
DatagramSocket mySocket1 =
new DatagramSocket(5775);
application
P3
transport
network
link
physical
application
P1
transport
network
link
physical
application
P4
transport
network
link
physical
source port: 6428
dest port: 9157
source port: ?
dest port: ?
source port: 9157
dest port: 6428
source port: ?
dest port: ?

Exemplo


DatagramSocket serverSocket =
new DatagramSocket (6428);
DatagramSocket mySocket2 =
new DatagramSocket(9157);
DatagramSocket mySocket1 =
new DatagramSocket(5775);
application
P3
transport
network
link
physical
application
P1
transport
network
link
physical
application
P4
transport
network
link
physical
source port: 6428
dest port: 9157
source port: ?
dest port: ?
source port: 9157
dest port: 6428
source port: 5775
dest port: 6428

Exemplo

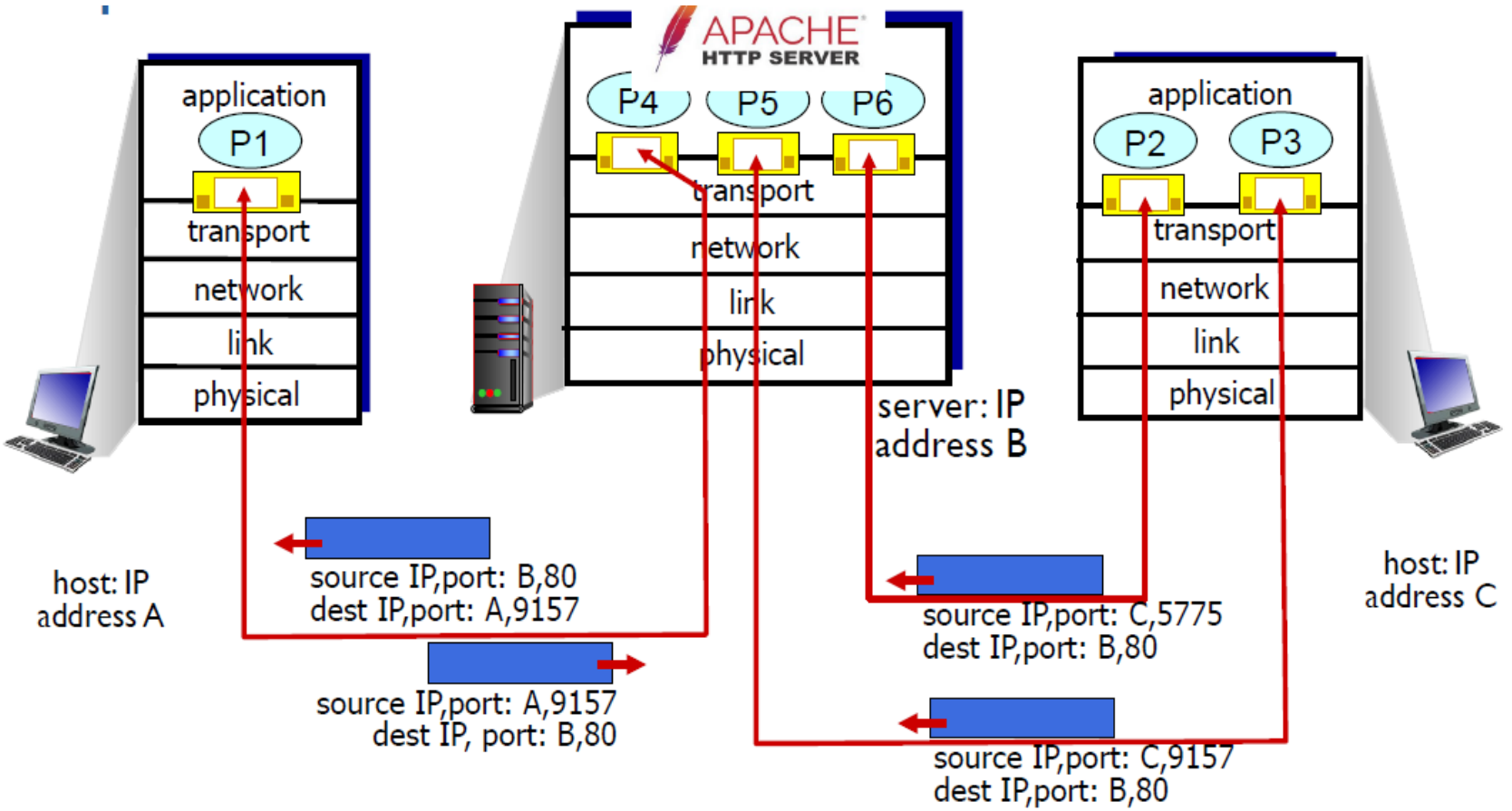

DatagramSocket serverSocket =
new DatagramSocket (6428);
DatagramSocket mySocket2 =
new DatagramSocket(9157);
DatagramSocket mySocket1 =
new DatagramSocket(5775);
application
P3
transport
network
link
physical
application
P1
transport
network
link
physical
application
P4
transport
network
link
physical
source port: 6428
dest port: 9157
source port: 6428
dest port: 5775
source port: 9157
dest port: 6428
source port: 5775
dest port: 6428

Demultiplexação orientada a conexões

- Socket TCP identificado pela quádrupla:
  - endereço IP origem
  - número da porta origem
  - endereço IP destino
  - número da porta destino
- Demultiplexação: receptor usa todos os quatro valores para direcionar o segmento para o socket apropriado

- Servidor pode dar suporte a muitos sockets TCP simultâneos:
  - cada socket é identificado pela sua própria quádrupla
- Servidores Web têm sockets diferentes para cada conexão de cliente
  - HTTP não persistente terá sockets diferentes para cada pedido

Exemplo



Três segmentos, todos destinados ao endereço IP B,
dest port: 80 são demultiplexados para sockets distintos

Redes de Computadores

Capítulo 3: Camada de transporte

Transporte não orientado para conexão: UDP

UDP: User Datagram Protocol

- Protocolo de transporte da Internet mínimo, “sem gorduras”,
- Serviço “melhor esforço”, segmentos UDP podem ser:
  - perdidos
  - entregues à aplicação for a de ordem
- sem conexão:
  - não há saudação inicial entre o remetente e o receptor UDP
- tratamento independente para cada segmento UDP

UDP: User Datagram Protocol

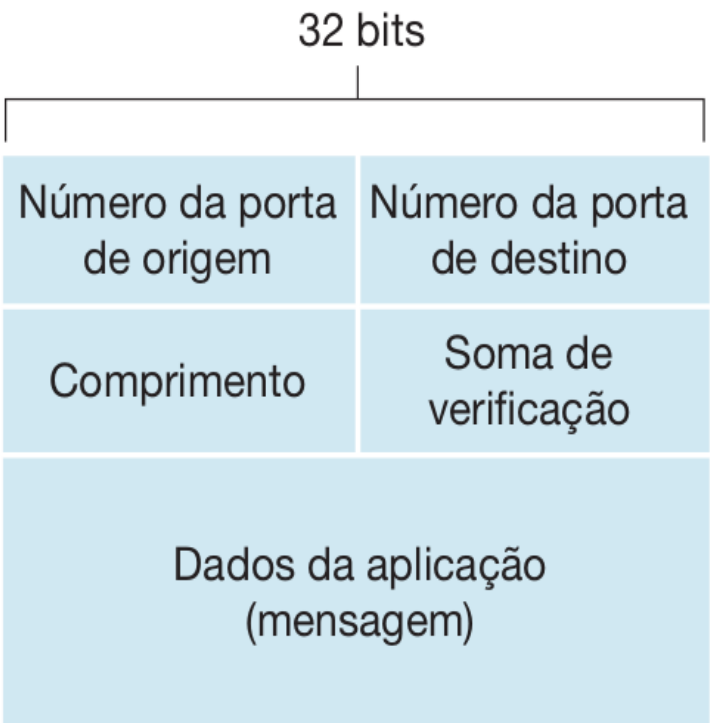
- Uso do UDP
  - aplicações de streaming multimídia (tolerante a perdas, sensível a taxas)
  - DNS
  - SNMP
- transferência confiável sobre UDP:
  - adiciona confiabilidade na camada de aplicação
  - recuperação de erros específica da aplicação

UDP

- Elimina estabelecimento de conexão (que pode causar retardo)
- Simples: não mantém “estado” da conexão nem no remetente, nem no receptor
- Cabeçalho de segmento reduzido
- Não há controle de congestionamento: UDP pode transmitir tão rápido quanto desejado (e possível)

Formado do segmento UDP

- Comprimento
  - Comprimento em bytes do segmento UDP incluindo o cabeçalho
  -
- Soma de verificação
  - Detectar “erros” no segmento transmitido
  - Erros: bits trocados



Soma de verificação do UDP

Transmissor:

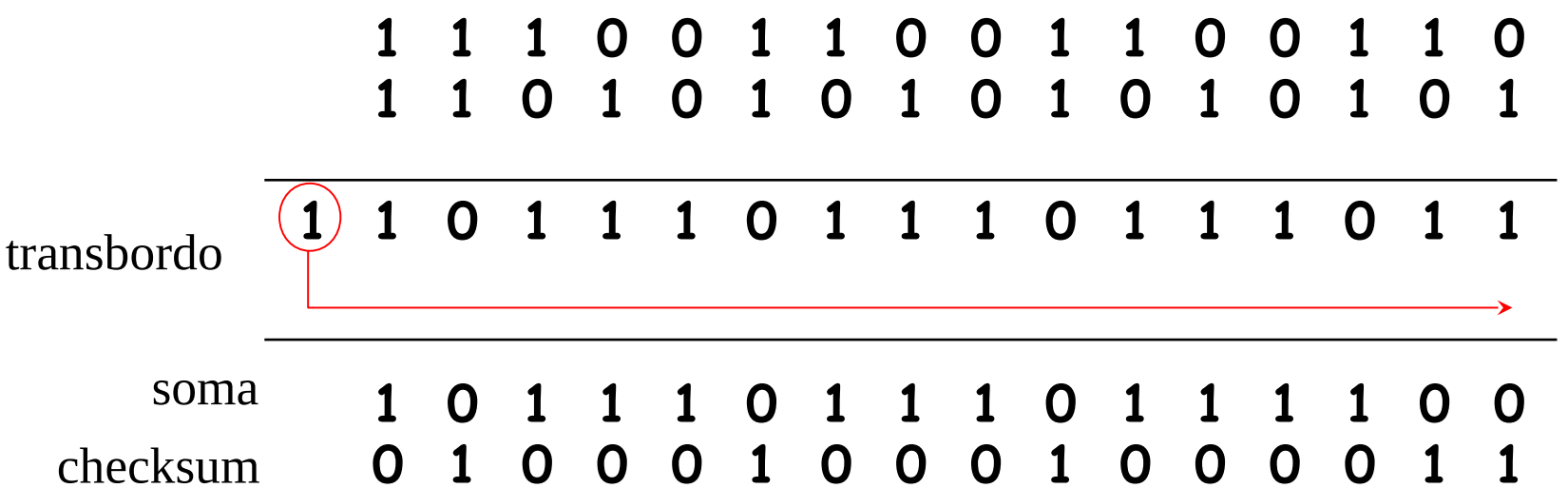
- Trata o conteúdo do segmento como sequência de inteiros de 16 bits
- checksum: soma (adição usando complemento de 1) do conteúdo do segmento
- Transmissor coloca complemento do valor da soma no campo checksum do UDP

Receptor:

- Calcula checksum do segmento recebido
- Verifica se o checksum calculado bate com o valor recebido:
  - NÃO - erro detectado
  - SIM - nenhum erro detectado. Mas ainda pode ter erros?

Soma de verificação do UDP

- Exemplo: adição de 2 número de 16 bits



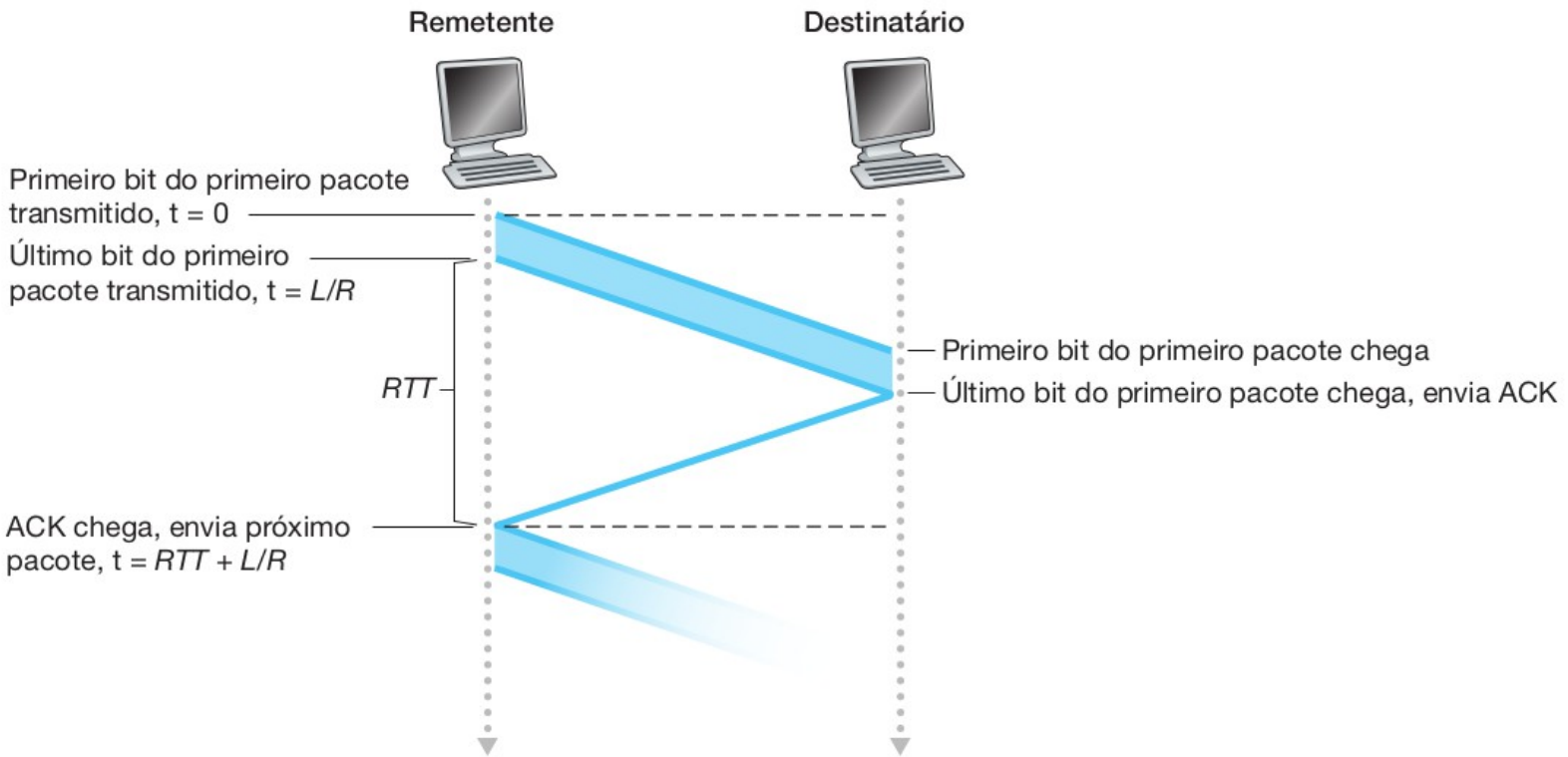
Note que: ao adicionar números, o transbordo (vai um) do bit mais significativo deve ser adicionado ao resultado

Redes de Computadores

Capítulo 3: Camada de transporte

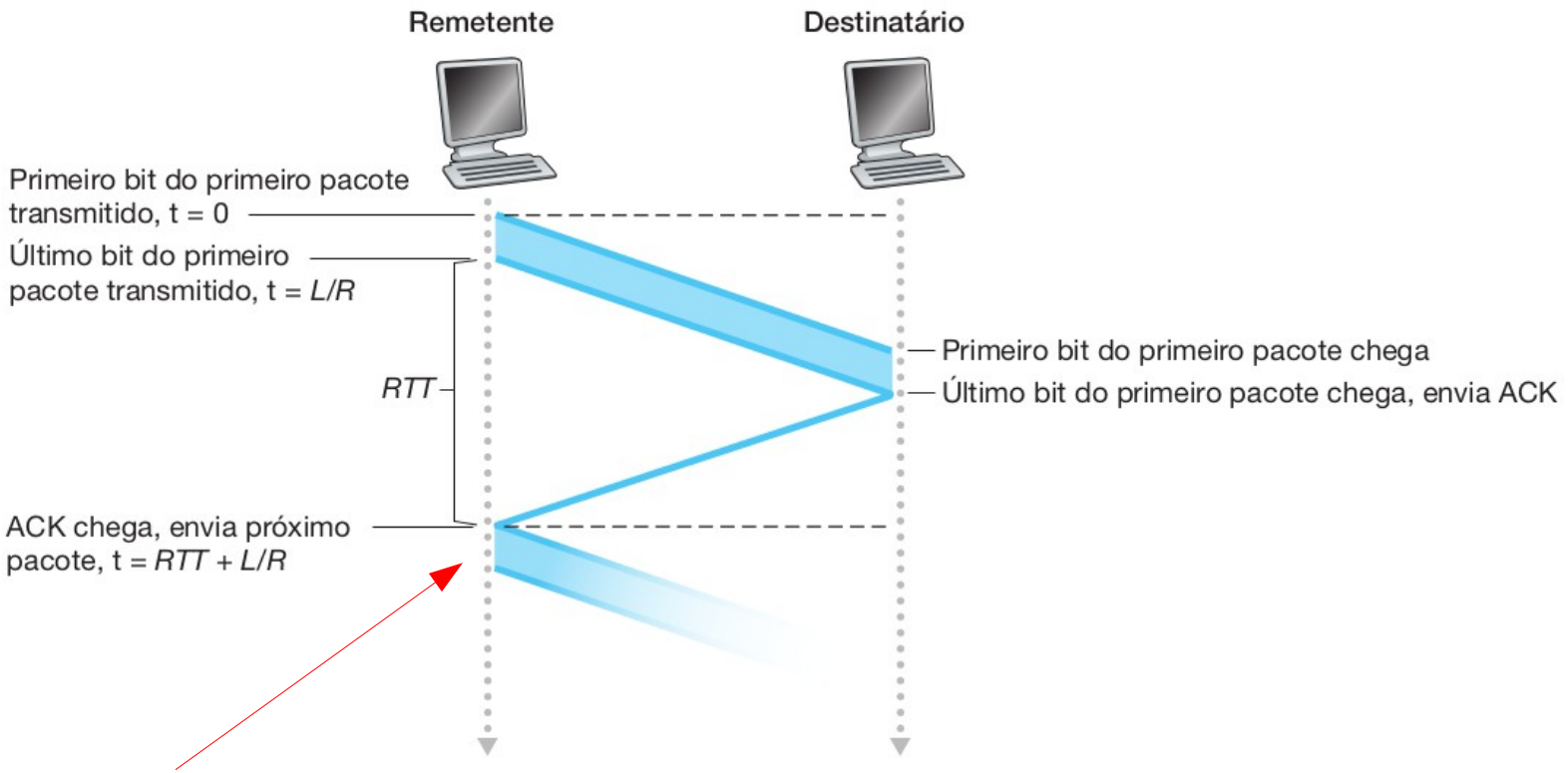
Introdução a comunicação por fluxo

Comunicação com pare e espere



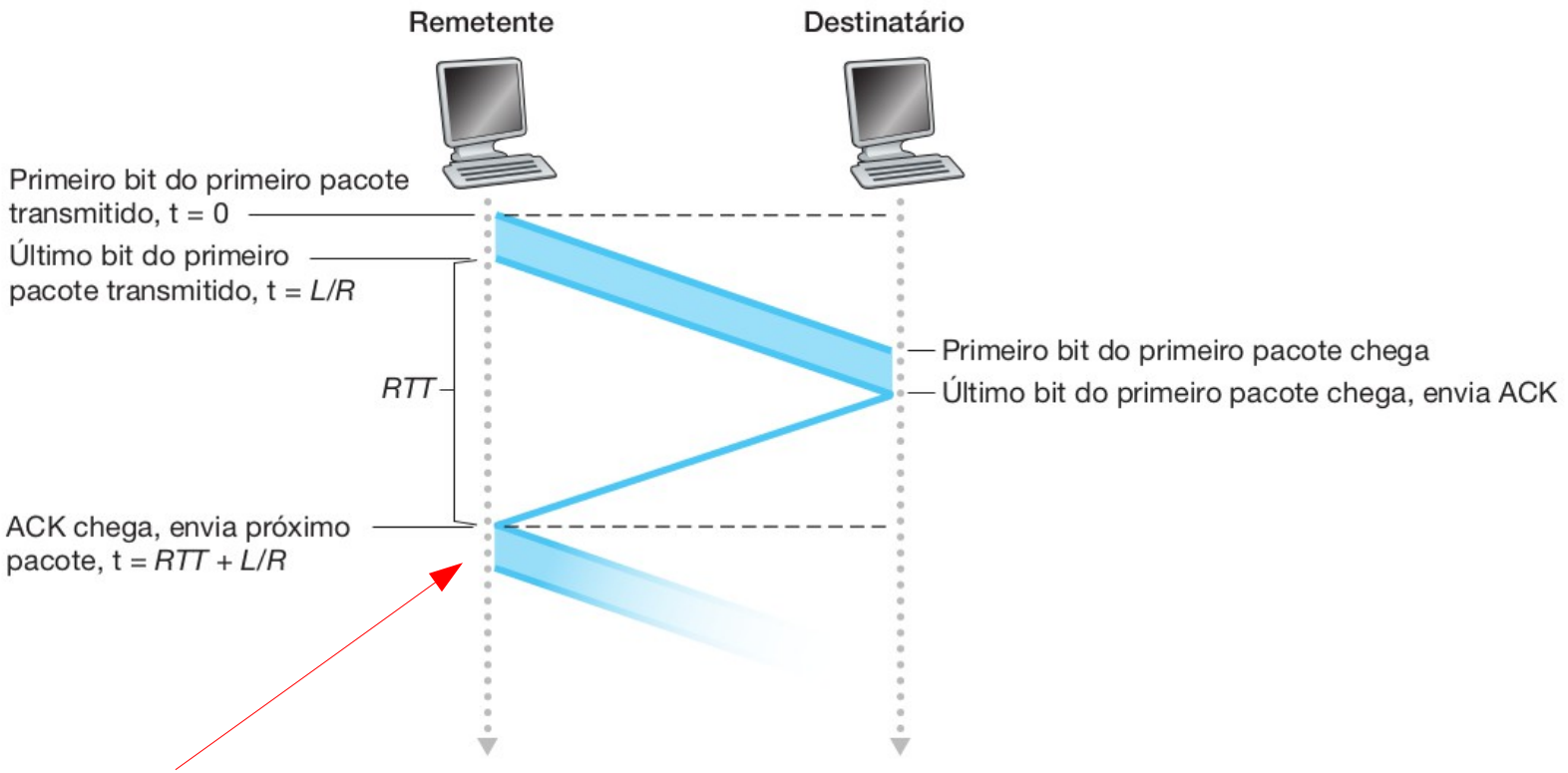
RTT: tempo de ida + volta

Comunicação com pare e espere



O segundo pacote é enviado após a confirmação de que o primeiro foi entregue

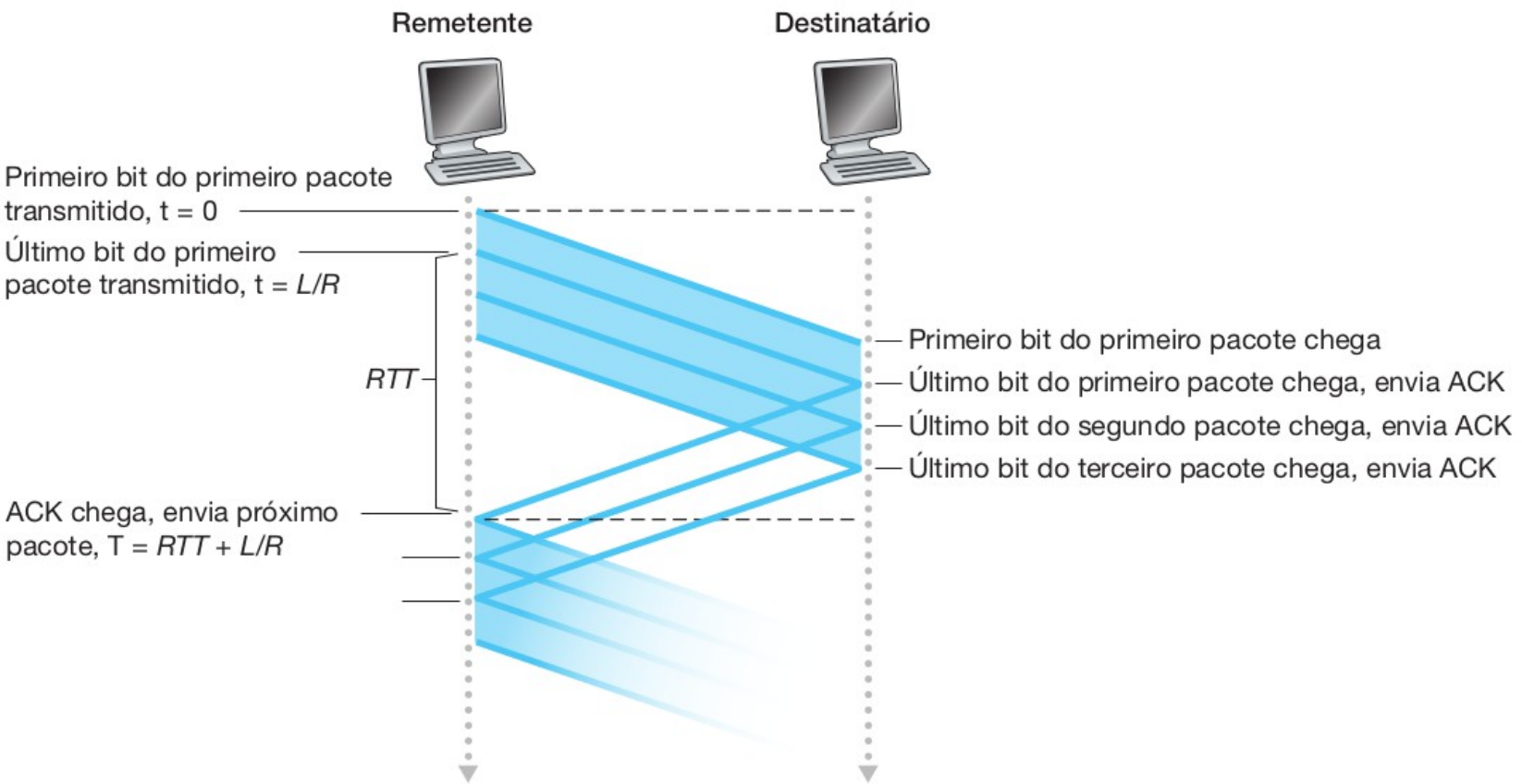
Comunicação com pare e espere



O segundo pacote é enviado após a confirmação de que o primeiro foi entregue

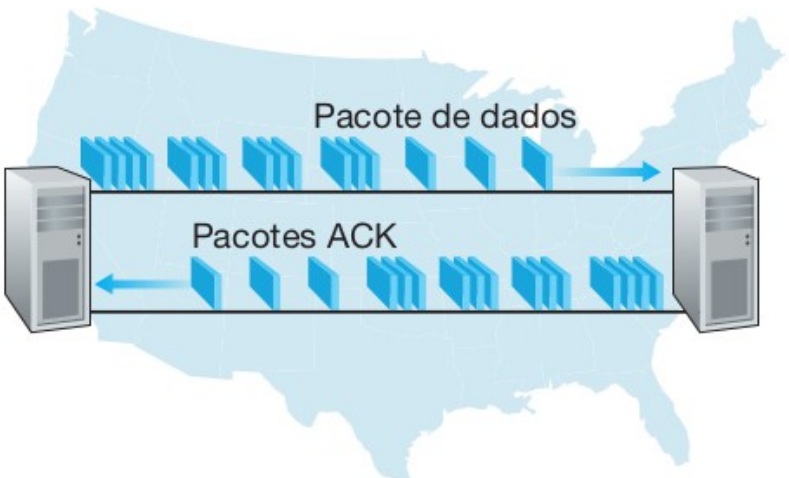
Problema? E se a confirmação ACK não chegou no remetente?

Comunicação com paralelismo



Comparação entre os protocolos

a. Um protocolo pare e espere em operação



b. Um protocolo com paralelismo em operação

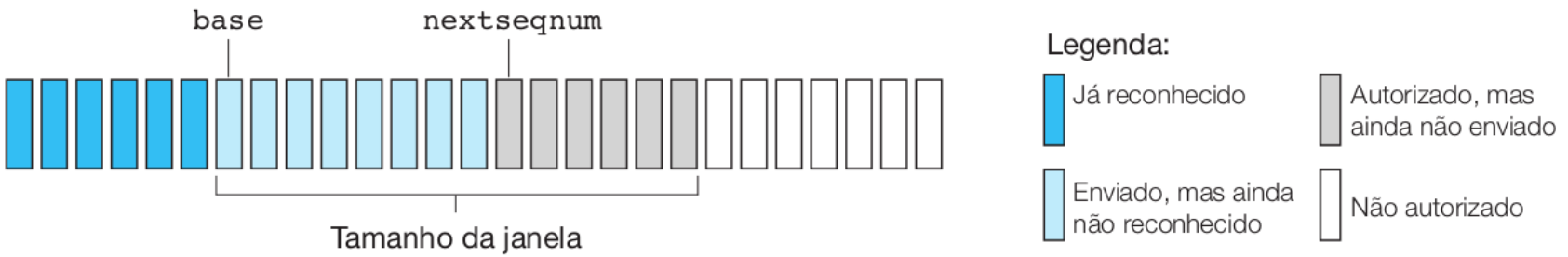
Protocolos com paralelismo

- Go-back-N
- Retransmissão seletiva

Go-back-N

Transmissor

- Número de sequência de k-bits no cabeçalho do pacote
- Admite “janela” de até N pacotes consecutivos não reconhecidos



ACK(n): reconhece todos pacotes, até e inclusive no. de seq n - “ACK/reconhecimento cumulativo”

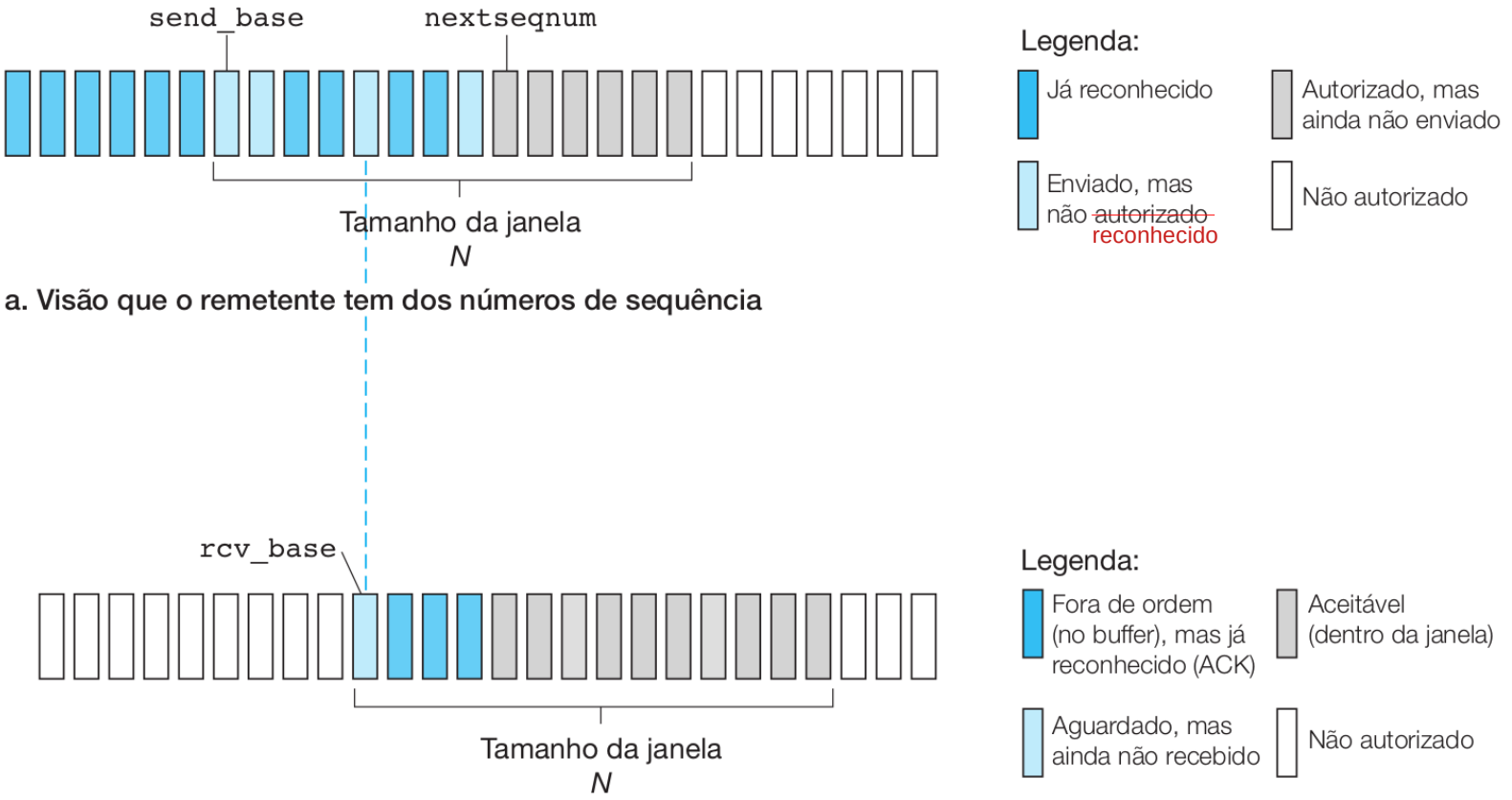
  - pode receber ACKs duplicados

- Temporizador para o pacote mais antigo ainda não confirmado
- Estouro do temporizador: retransmite todos os pacotes pendentes.

Retransmissão seletiva

- Receptor reconhece individualmente todos os pacotes recebidos corretamente
  - armazena pacotes no buffer, conforme necessário, para posterior entrega em-ordem à camada superior
- Transmissor apenas reenvia pacotes para os quais um ACK não foi recebido
  - temporizador de remetente para cada pacote sem ACK
- Janela do transmissão
  - N números de sequência consecutivos
  - Outra vez limita números de sequência de pacotes enviados, mas ainda não reconhecidos

Retransmissão seletiva: janelas do transmissor e receptor



Redes de Computadores

Capítulo 3: Camada de transporte

Aula 05: Transporte orientado para conexão: TCP

TCP: visão geral

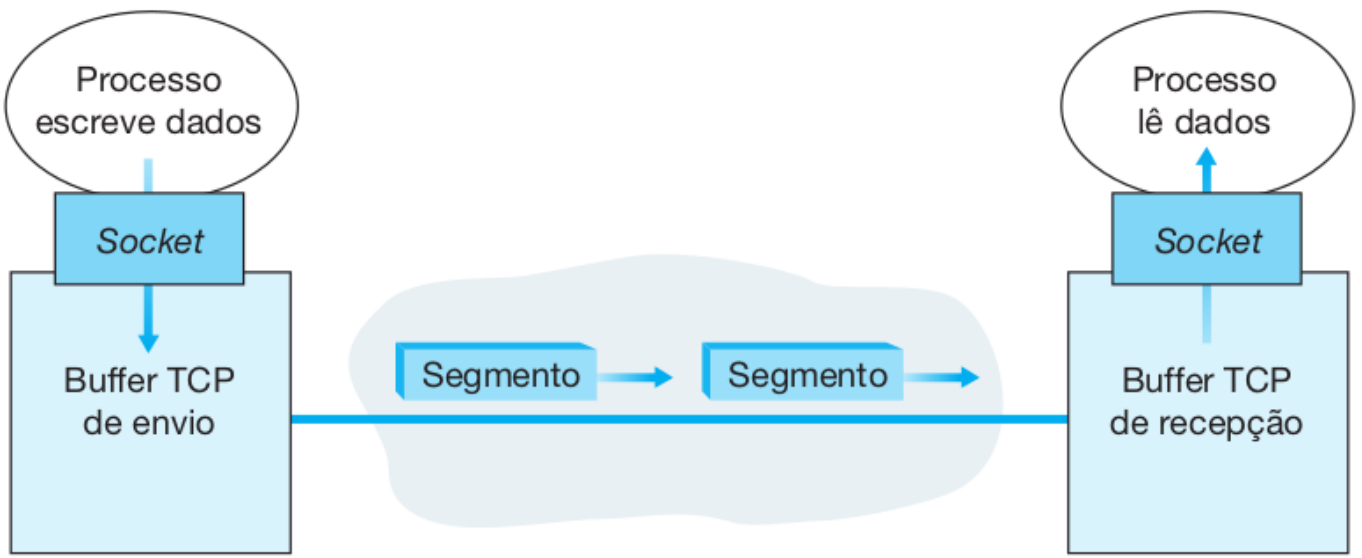
- Ponto a ponto
  - Um transmissor e um receptor
- Fluxo de bytes, ordenados e confiável
  - Recepção da mensagem pode ser feita fora de ordem, mas entrega em ordem para a aplicação
- Paralelismo
  - Utiliza o conceito de janela, que é ajustada pelo controle de fluxo e controle de congestionamento

TCP: visão geral

- Transmissão full-duplex
  - Fluxo de dados bi-direcional na mesma coneão
  - MSS: tamanho máximo de um segmento
- Orientado a conexão
  - Handshaking (troca de msgs de controle)
  - Inicia estado do transmissor e do receptor antes da troca de dados
- Fluxo controlado
  - O receptor não será afogado pelo transmissor

TCP: visão geral

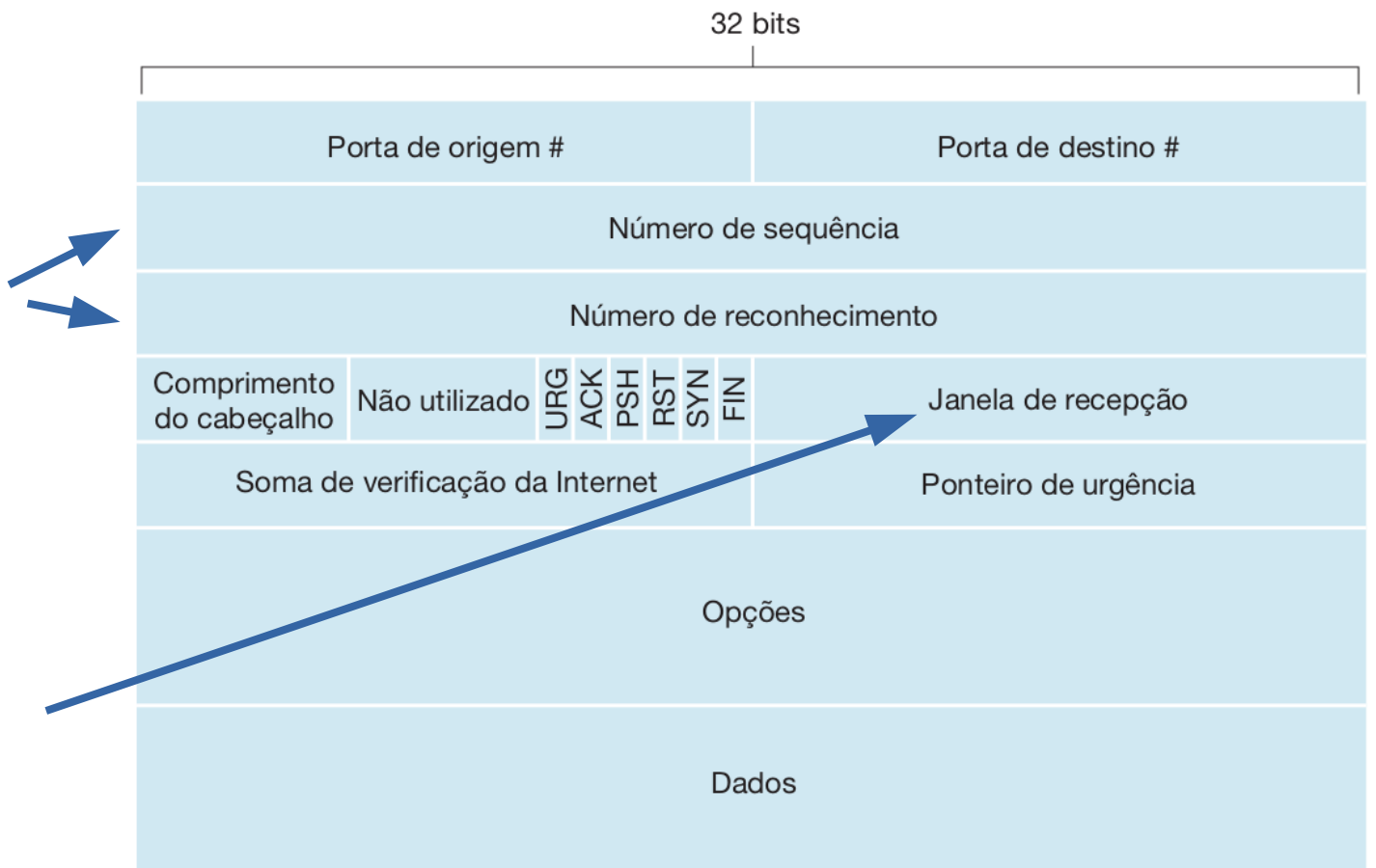
- *buffers* de envio e recepção



Estrutura do segmento TCP

Contagem por bytes de dados e não segmentos

Número de bytes que o receptor está pronto para receber



Estrutura do segmento TCP

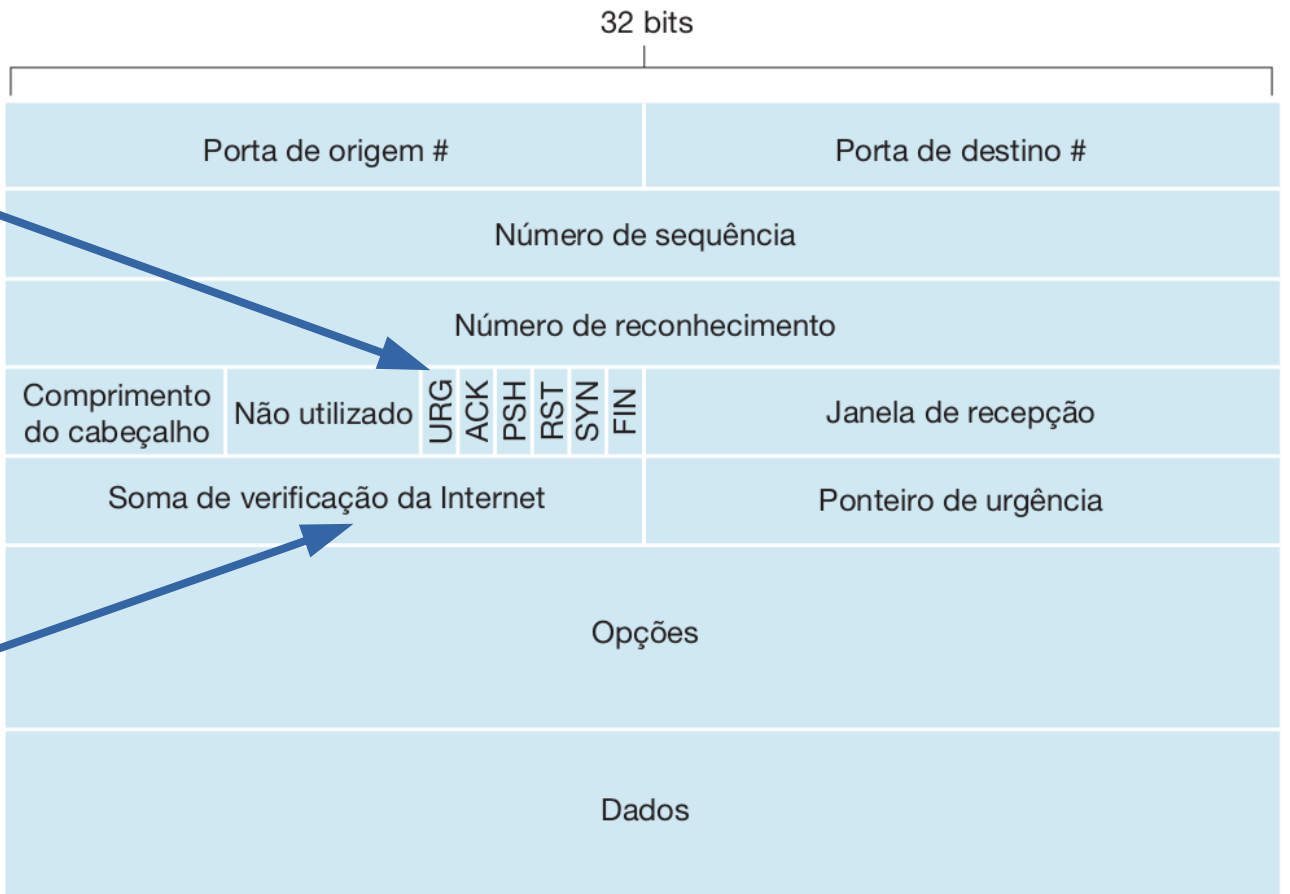
URG: dados urgentes (pouco usado)

ACK: campo de reconhecimento é valido

PSH: produz o envio de dados (pouco utilizado)

RST, SYN, FIN: estabelecimento e encerramento de conexão

Como no UDP



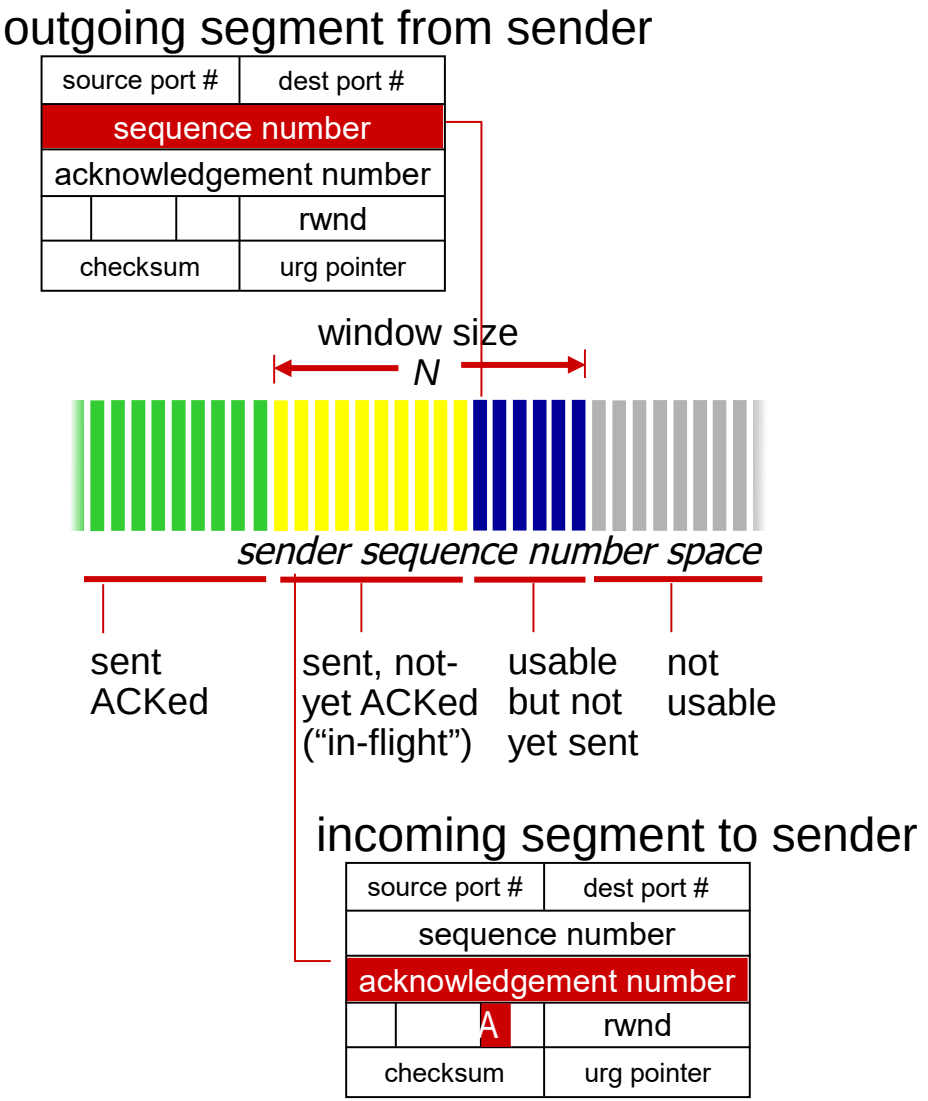
TCP: número de sequência e ACKs

Números de sequência:

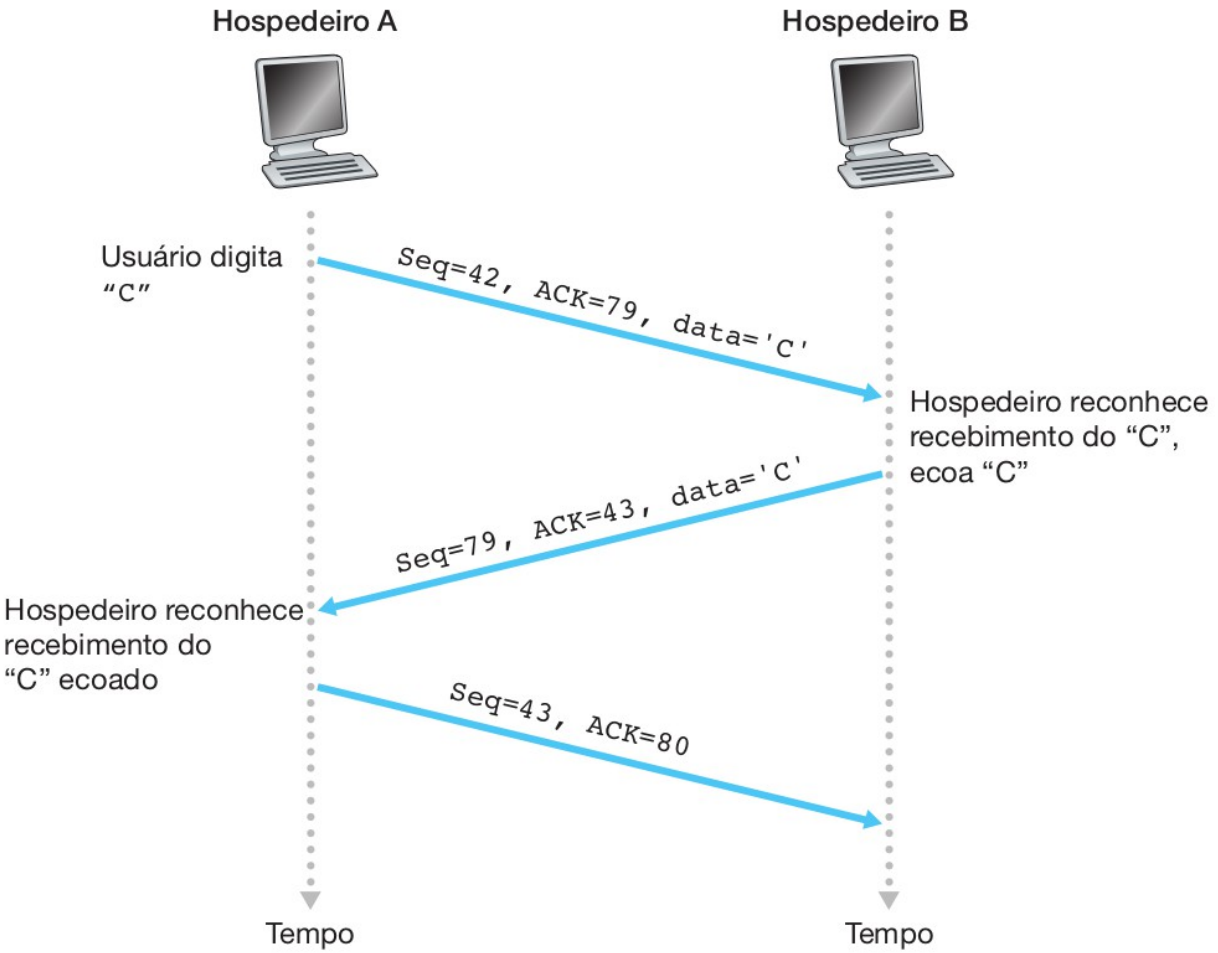
- "número" dentro do fluxo de bytes do primeiro byte de dados do segmento

ACKs:

- número de seq. do próx. Byte esperado do outro lado
- ACK cumulativo



TCP: número de sequência e ACKs



Cenário telnet

Redes de Computadores

Capítulo 3: Camada de transporte

Transporte orientado para conexão: TCP

TCP: visão geral

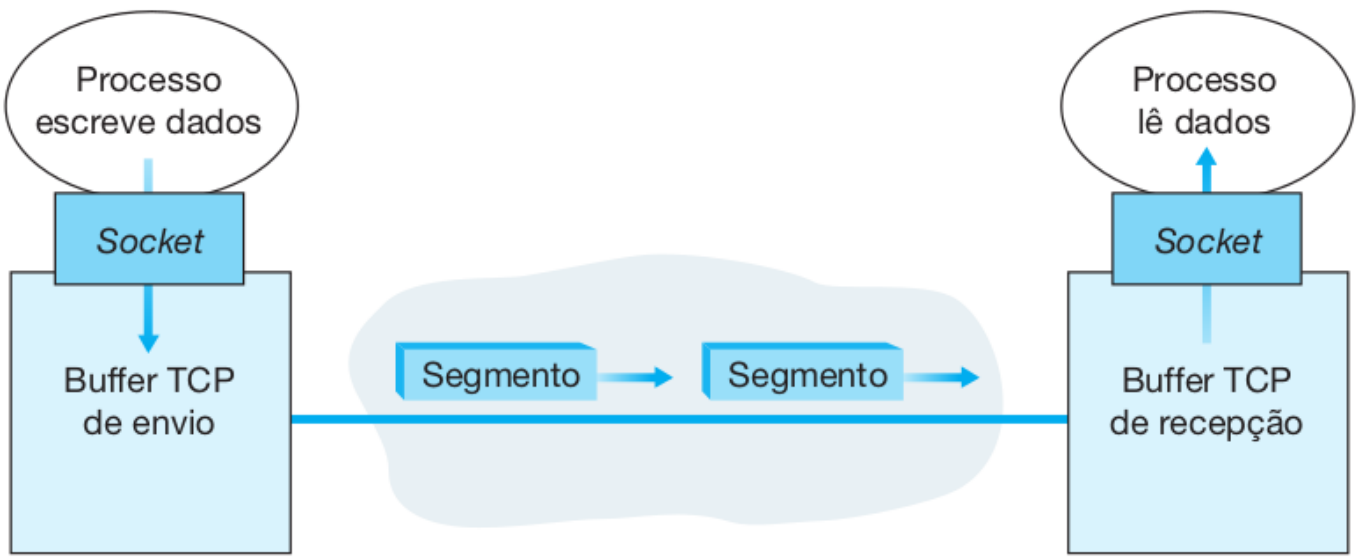
- Ponto a ponto
  - Um transmissor e um receptor
- Fluxo de bytes, ordenados e confiável
  - Recepção da mensagem pode ser feita fora de ordem, mas entrega em ordem para a aplicação
- Paralelismo
  - Utiliza o conceito de janela, que é ajustada pelo controle de fluxo e controle de congestionamento

TCP: visão geral

- Transmissão full-duplex
  - Fluxo de dados bi-direcional na mesma coneão
  - MSS: tamanho máximo de um segmento
- Orientado a conexão
  - Handshaking (troca de msgs de controle)
  - Inicia estado do transmissor e do receptor antes da troca de dados
- Fluxo controlado
  - O receptor não será afogado pelo transmissor

TCP: visão geral

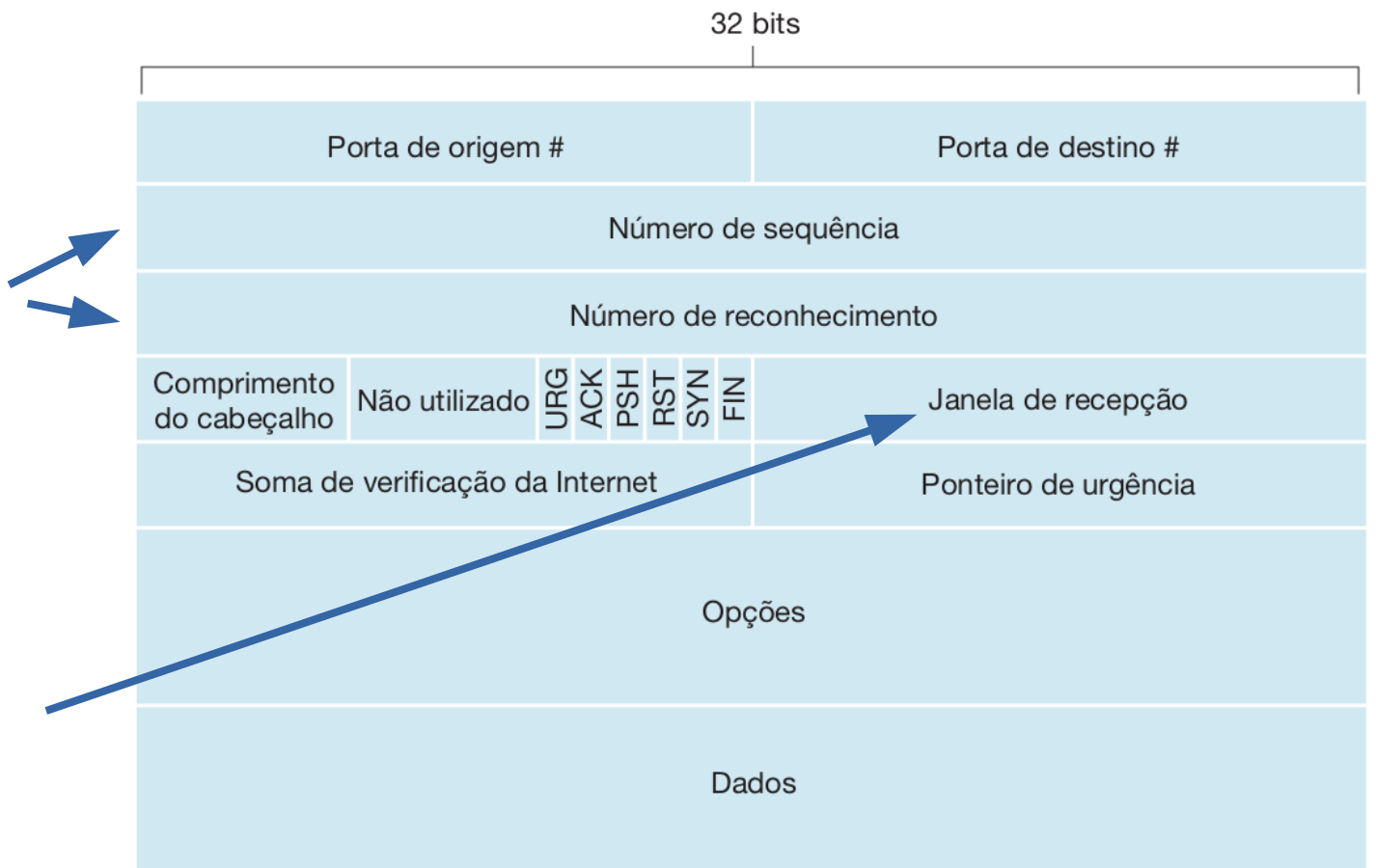
- *buffers* de envio e recepção



Estrutura do segmento TCP

Contagem por bytes de dados e não segmentos

Número de bytes que o receptor está pronto para receber



Estrutura do segmento TCP

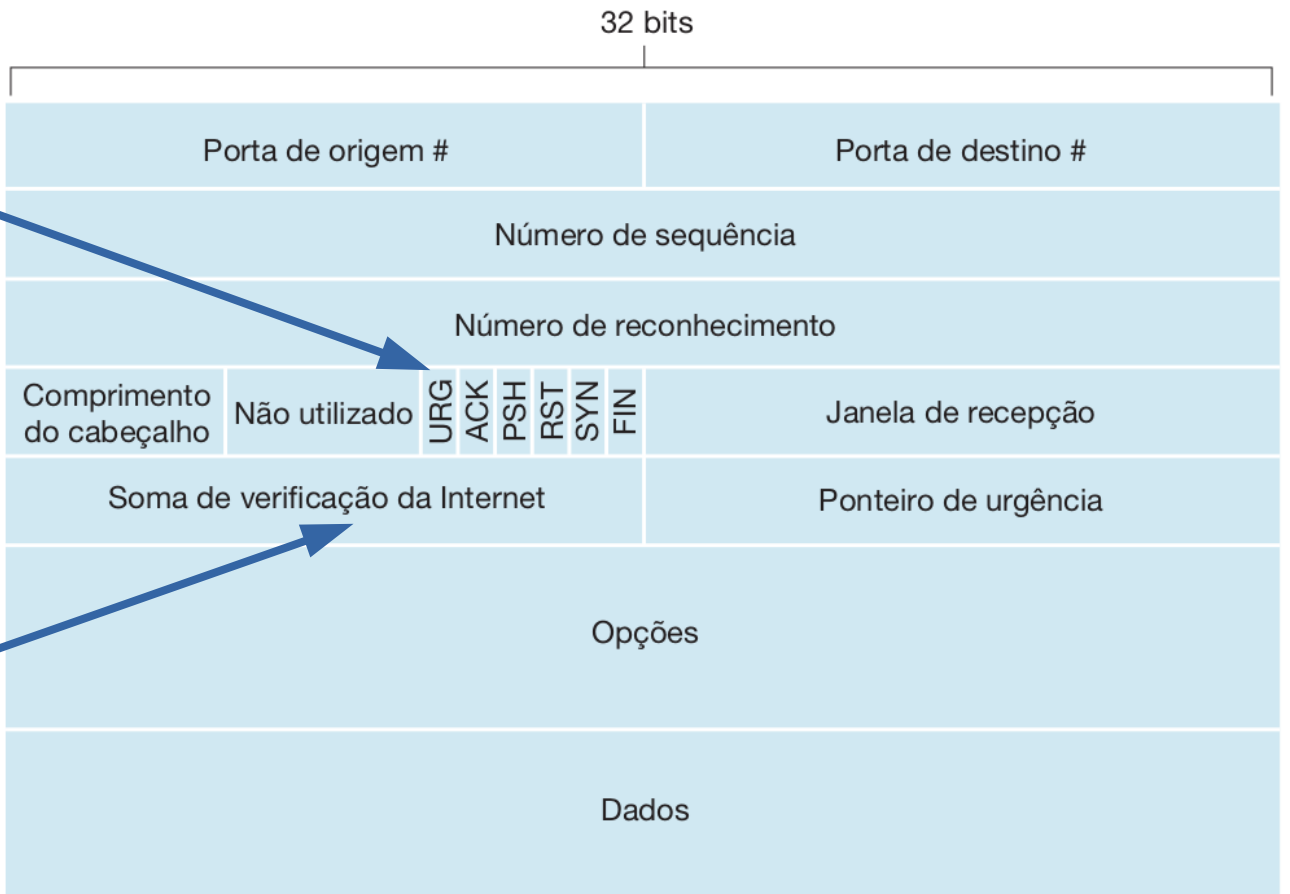
URG: dados urgentes (pouco usado)

ACK: campo de reconhecimento é valido

PSH: produz o envio de dados (pouco utilizado)

RST, SYN, FIN: estabelecimento e encerramento de conexão

Como no UDP



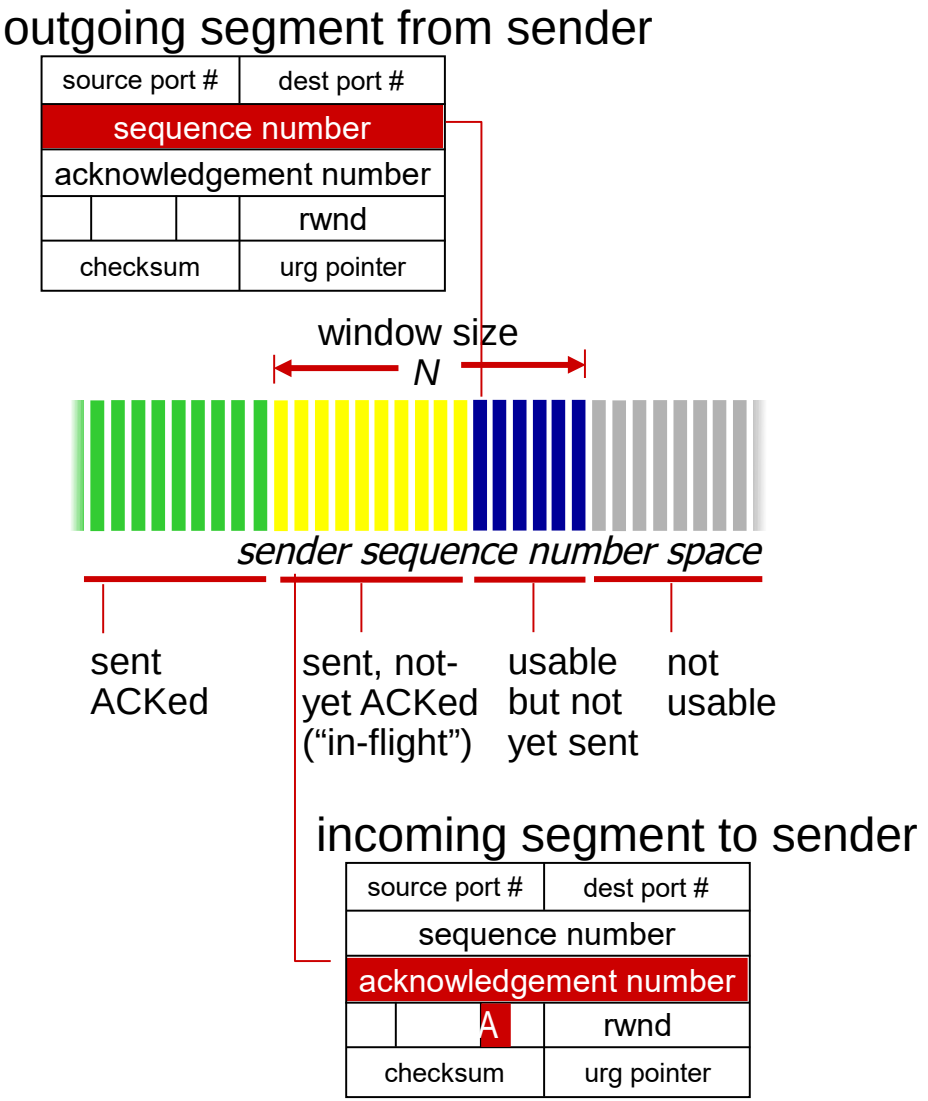
TCP: número de sequência e ACKs

Números de sequência:

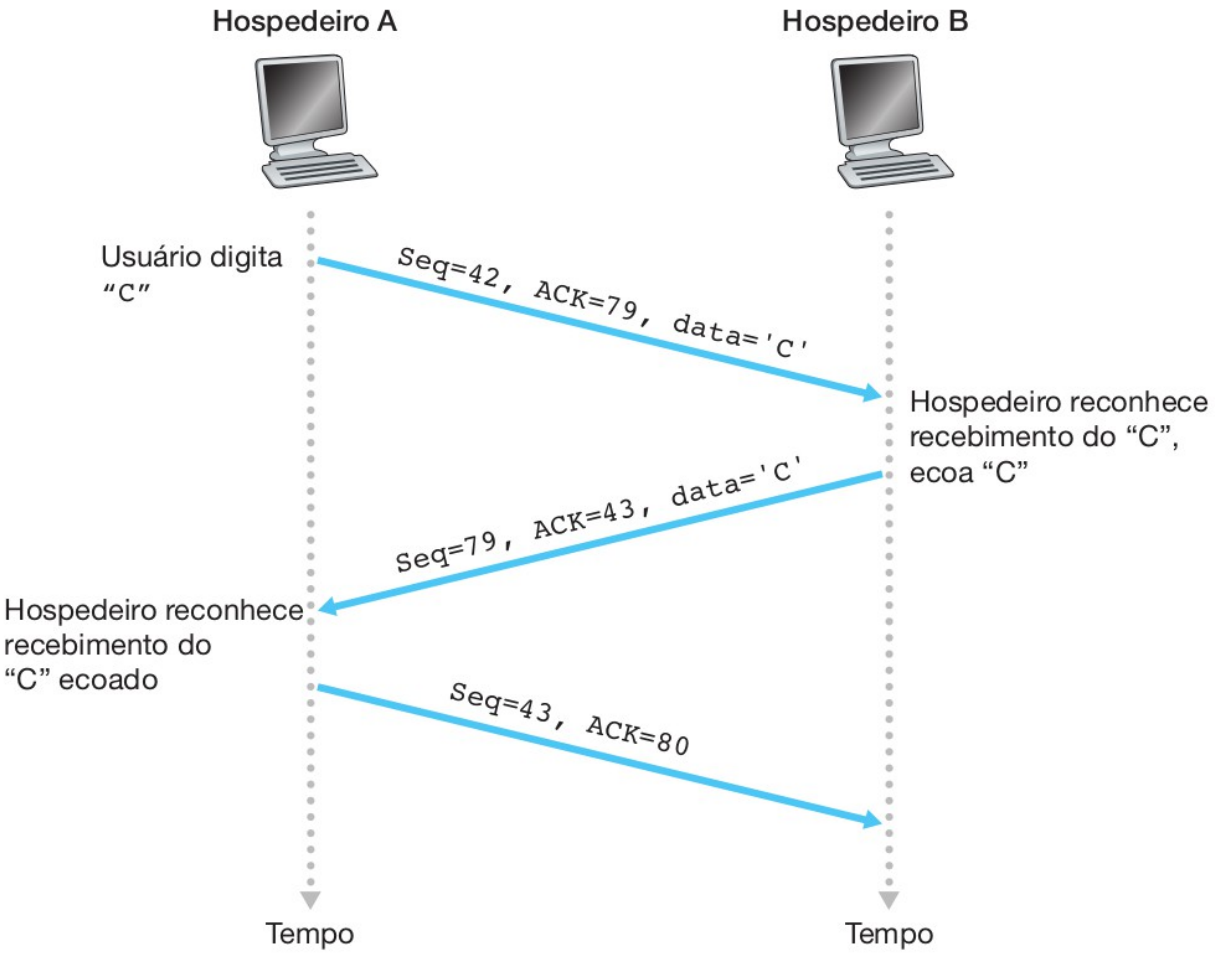
- "número" dentro do fluxo de bytes do primeiro byte de dados do segmento

ACKs:

- número de seq. do próx. Byte esperado do outro lado
- ACK cumulativo



TCP: número de sequência e ACKs



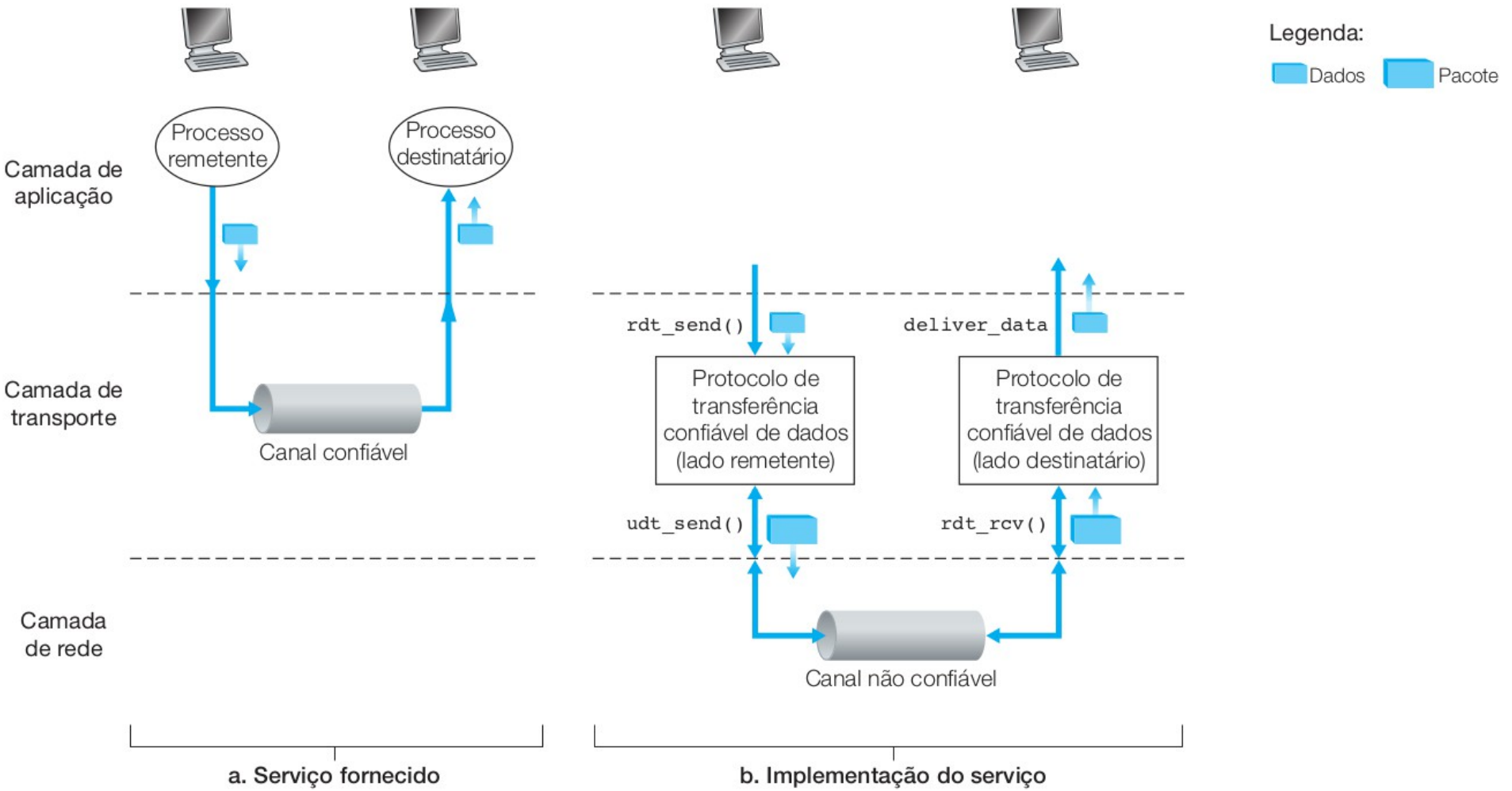
Cenário telnet

Redes de Computadores

Capítulo 3: Camada de transporte

Transporte orientado para conexão: TCP
continuação...

Transferência confiável de dados



Transferência confiável de dados

- O TCP cria um serviço rdt (realible data transfer) sobre o serviço não confiável do IP
  - Segmentos transmitidos em “paralelo” (pipelined)
  - Acks cumulativos
  - O TCP usa um único temporizador para retransmissões
- As retransmissões são disparadas por:
  - estouros de temporização
  - acks duplicados

Eventos no transmissor TCP

Dados recebidos da aplicação:

- Cria segmento com número de sequência (*nseq*)
- *nseq* é o número de sequência do primeiro byte de dados do segmento
- Liga o temporizador se já não estiver ligado (temporização do segmento mais antigo ainda não reconhecido)

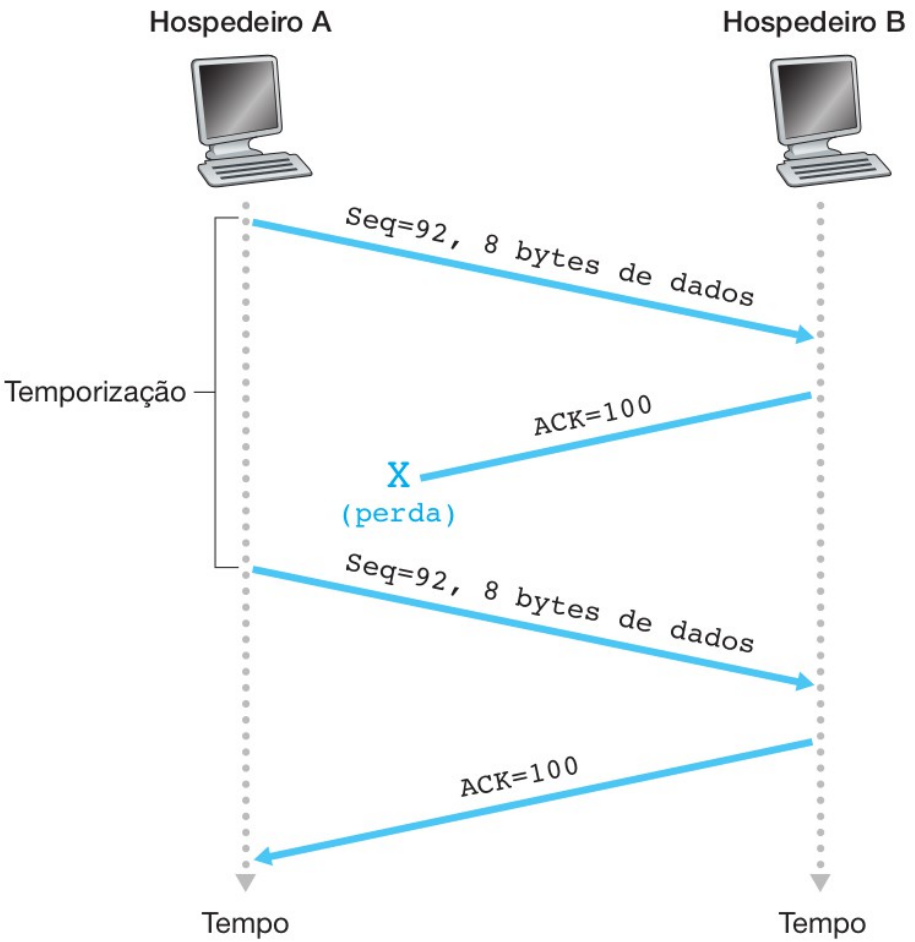
Estouro do temporizador

- Retransmite o segmento que causou o estouro do temporizador
- Reinicia o temporizador

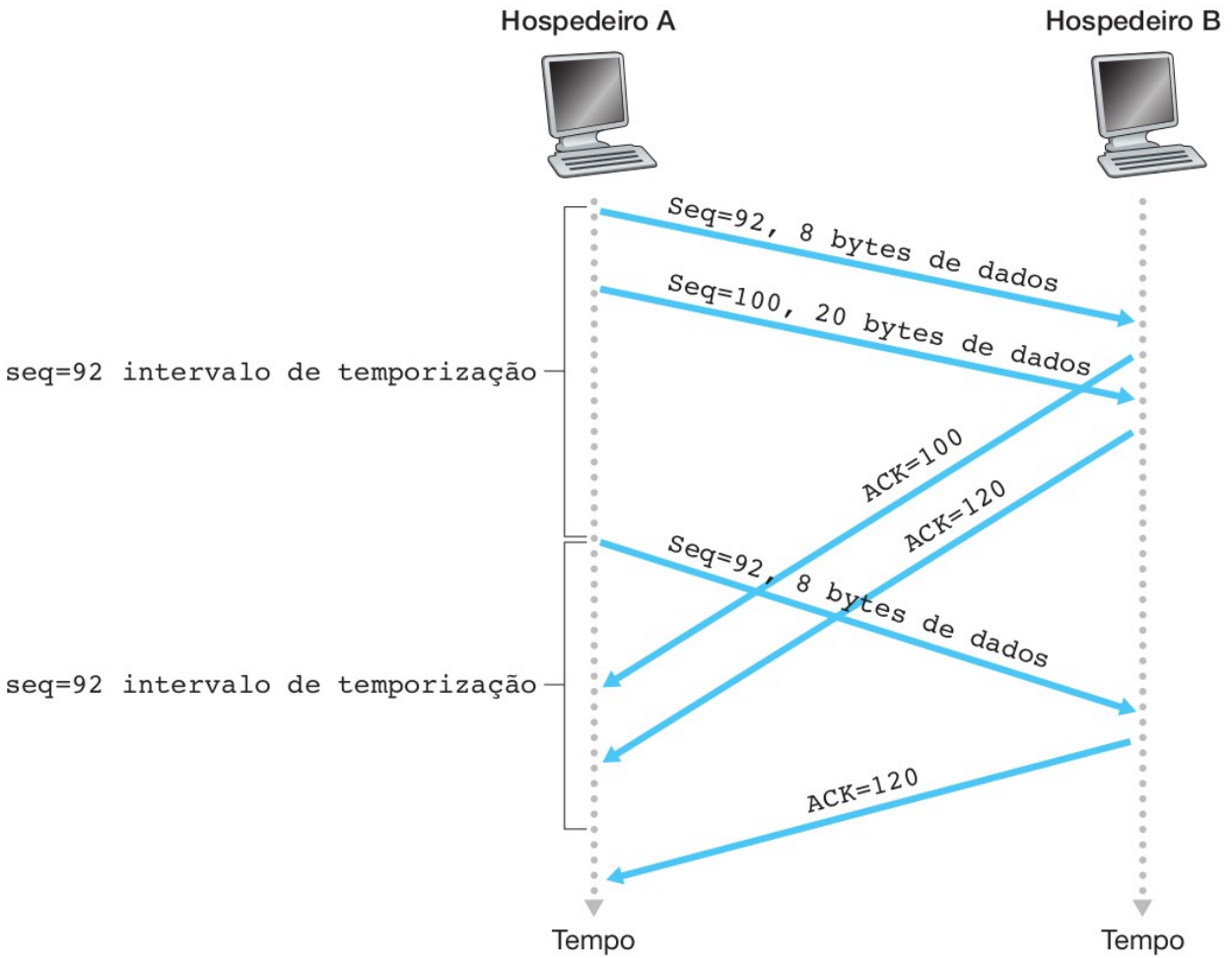
Recepção de Ack:

- Se reconhecer segmentos ainda não reconhecidos
  - atualizar informação sobre o que foi reconhecido
  - religa o temporizador se ainda houver segmentos pendentes (não reconhecidos)

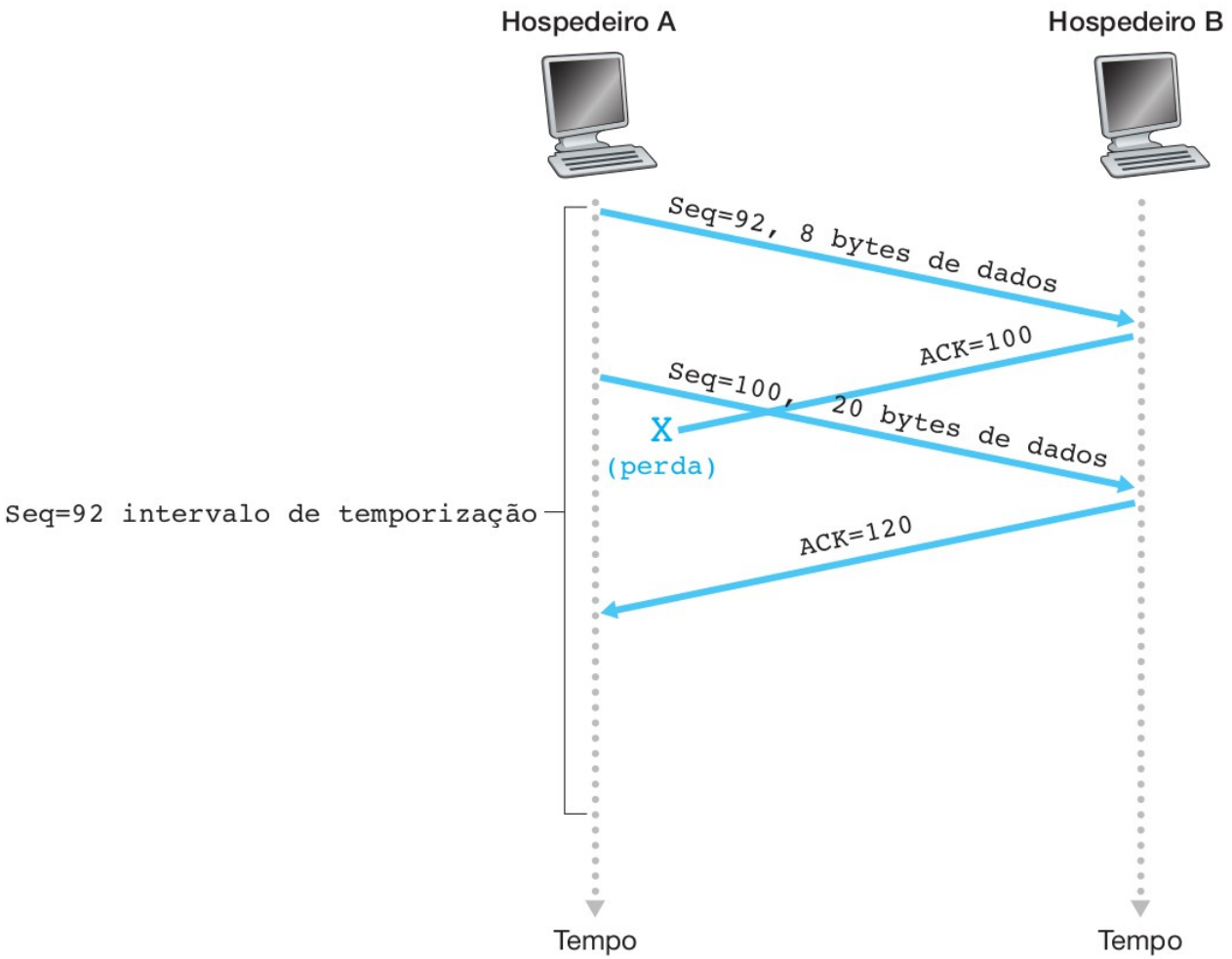
TCP: cenários de retransmissões



TCP: cenários de retransmissões



TCP: cenários de retransmissões



Geração de ACKs do TCP

Evento no recptor	Ação do Receptor TCP
chegada de segmento em ordem sem lacunas, anteriores já reconhecidos	ACK retardado. Espera até 500ms pelo próx. segmento. Se não chegar segmento, envia ACK
chegada de segmento em ordem sem lacunas, um ACK retardado pendente	envia imediatamente um único ACK cumulativo
chegada de segmento fora de ordem, com no. de seq. Maior que esperado -> lacuna	envia ACK duplicado, indicando no. de seq.do próximo byte esperado
chegada de segmento que preenche a lacuna parcial ou completamente	ACK imediato se segmento começa no início da lacuna

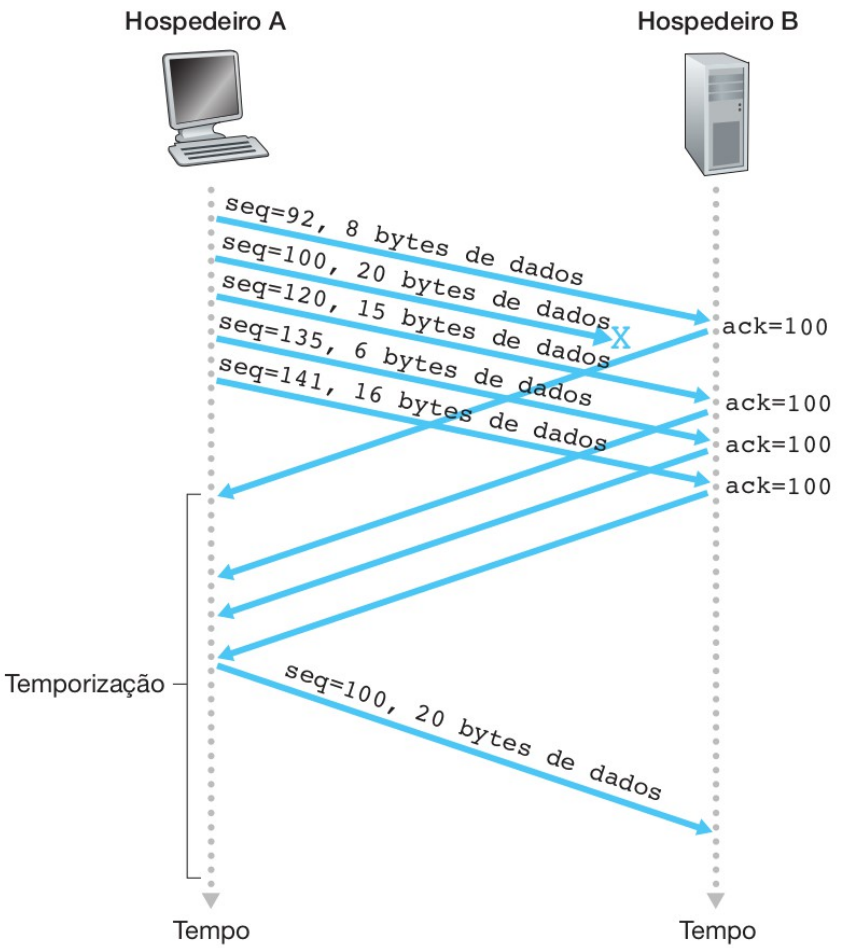
Retransmissão rápida do TCP

- O intervalo do temporizador é frequentemente bastante longo:
  - longo atraso antes de retransmitir um pacote perdido
- Detecta segmentos perdidos através de ACKs duplicados.
  - O transmissor normalmente envia diversos segmentos
  - Se um segmento se perder, provavelmente haverá muitos ACKs duplicados

Retransmissão rápida do TCP

- se o transmissor receber 3 ACKs adicionais para os mesmos dados (“três ACKs duplicados”), retransmite segmentos não reconhecidos com menores números de sequência
  - provavelmente o segmento não reconhecido se perdeu, não é preciso esperar o temporizador.

Retransmissão rápida do TCP



Redes de Computadores

Capítulo 3: Camada de transporte

Transporte orientado para conexão: TCP
continuação….

Controle de fluxo do TCP

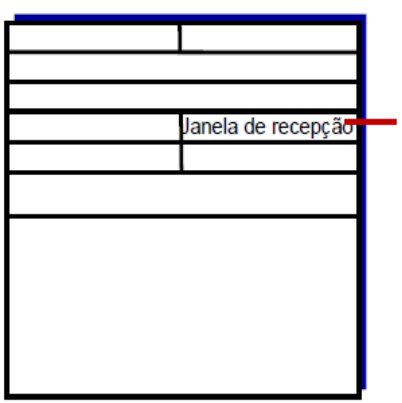
- Pergunta: o que acontece se a rede entregar dados em uma velocidade maior do que a aplicação pode consumir?

Controle de fluxo do TCP

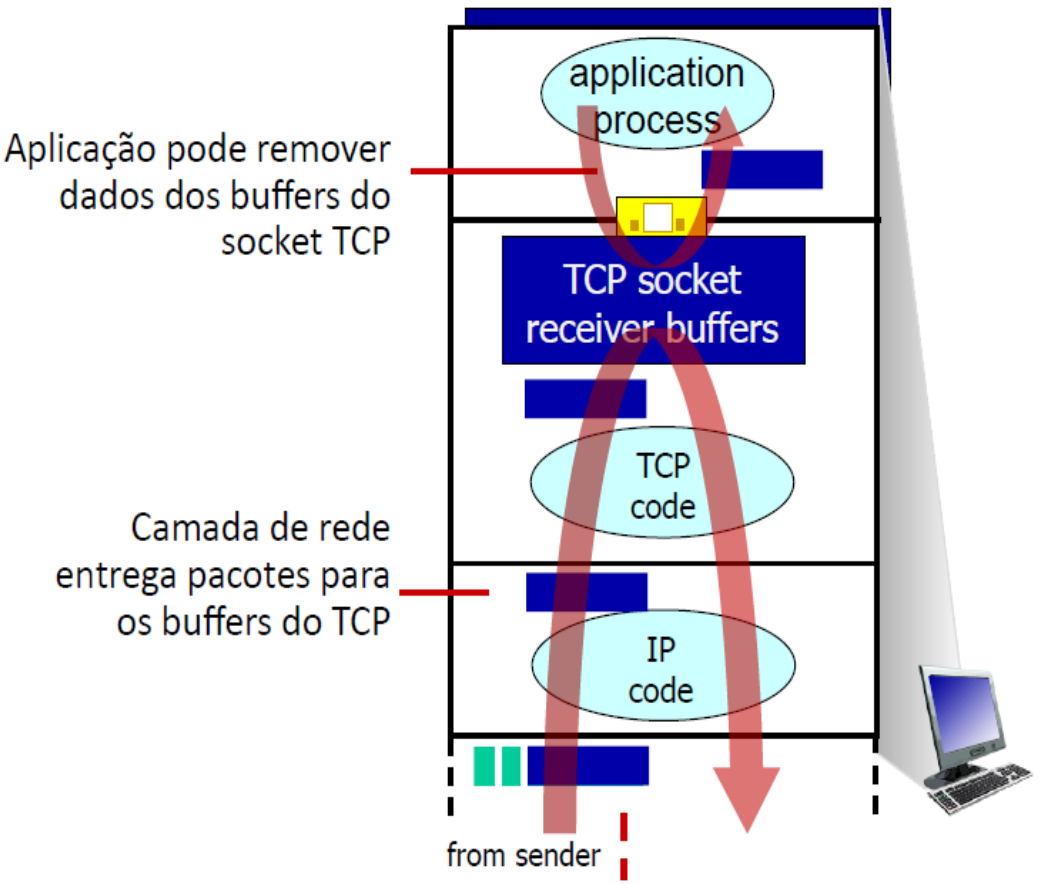
- Pergunta: o que acontece se a rede entregar dados em uma velocidade maior do que a aplicação pode consumir?

Controle de fluxo do TCP

- Pergunta: o que acontece se a rede entregar dados em uma velocidade maior do que a aplicação pode consumir?



Controle de fluxo: # de bytes que o receptor pode receber

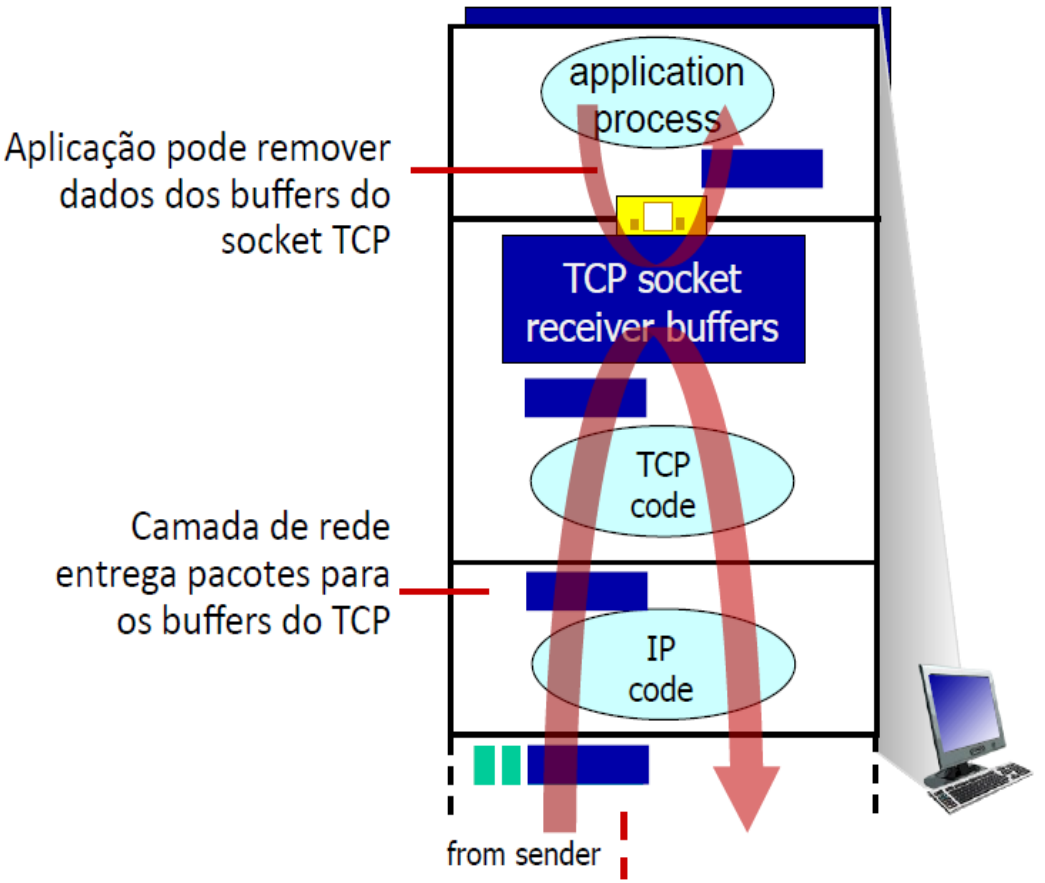


Pilha de protocolos no receptor

Controle de fluxo do TCP

- Pergunta: o que acontece se a rede entregar dados em uma velocidade maior do que a aplicação pode consumir?

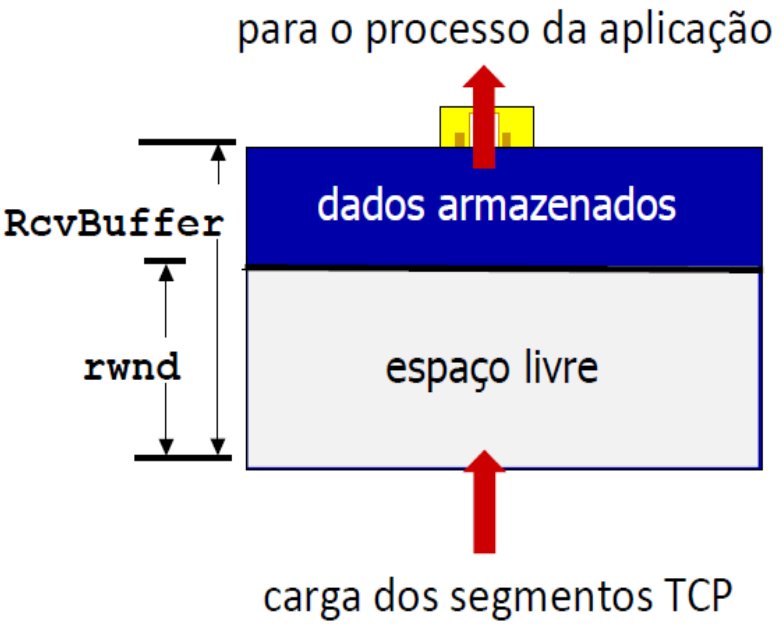
- Controle de fluxo: o receptor controla o transmissor, de modo que este não inunde o buffer do receptor transmitindo muito e rapidamente



Pilha de protocolos no receptor

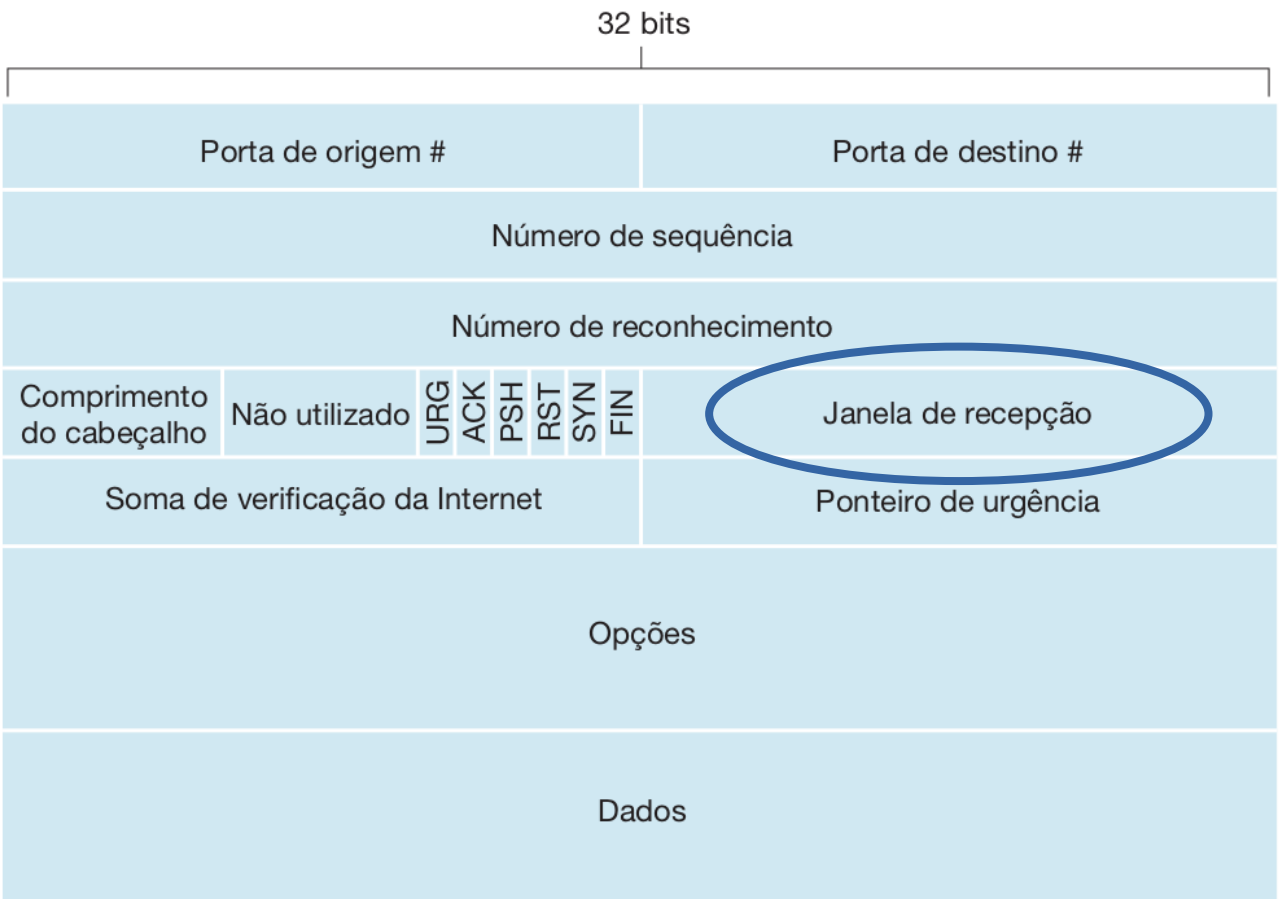
Controle de fluxo do TCP

- O receptor “anuncia” o espaço livre do buffer incluindo o valor da **rwnd** nos cabeçalhos TCP dos segmentos que saem do receptor para o transmissor
  - Tamanho do **RcvBuffer** é configurado através das opções do socket (o valor default é de 4096 bytes)
  - muitos sistemas operacionais ajustam **RcvBuffer** automaticamente.
- O transmissor limita a quantidade os dados não reconhecidos ao tamanho do **rwnd** recebido.
- Garante que o buffer do receptor não transbordará



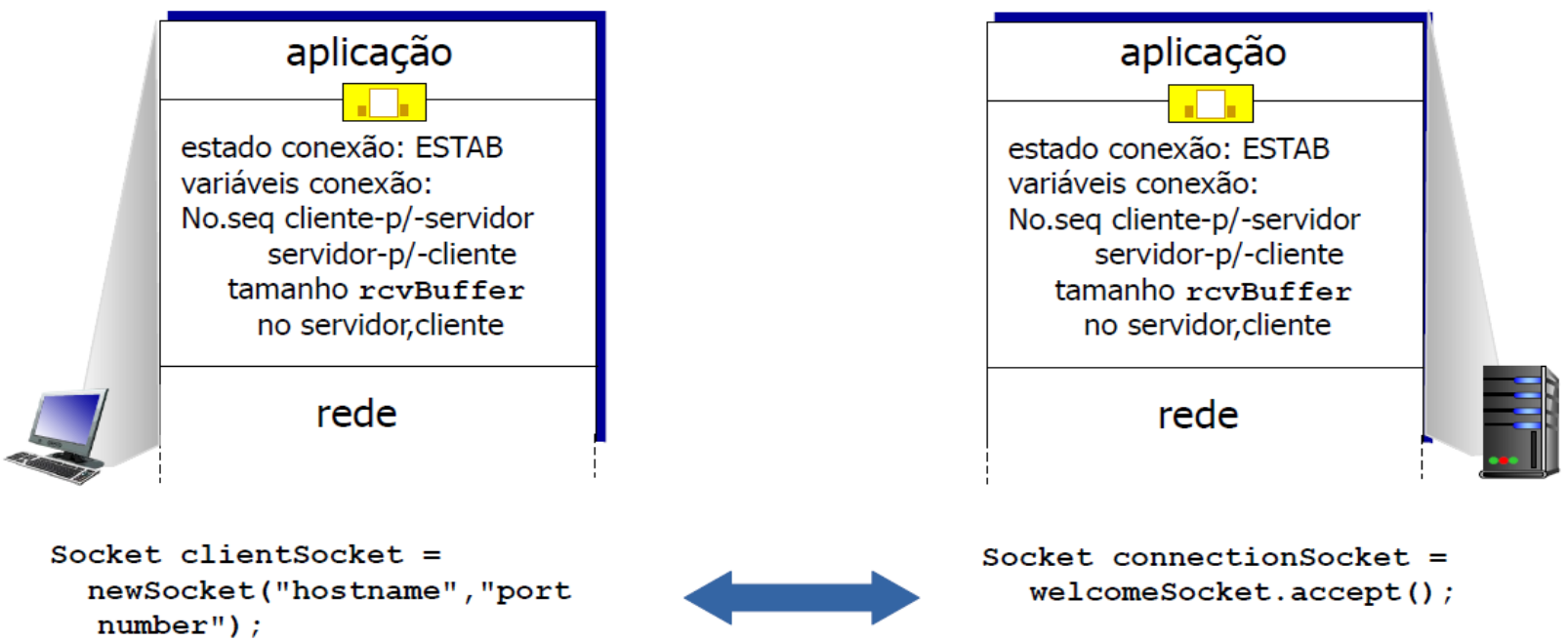
armazenamento no lado do receptor TCP

Controle de fluxo do TCP

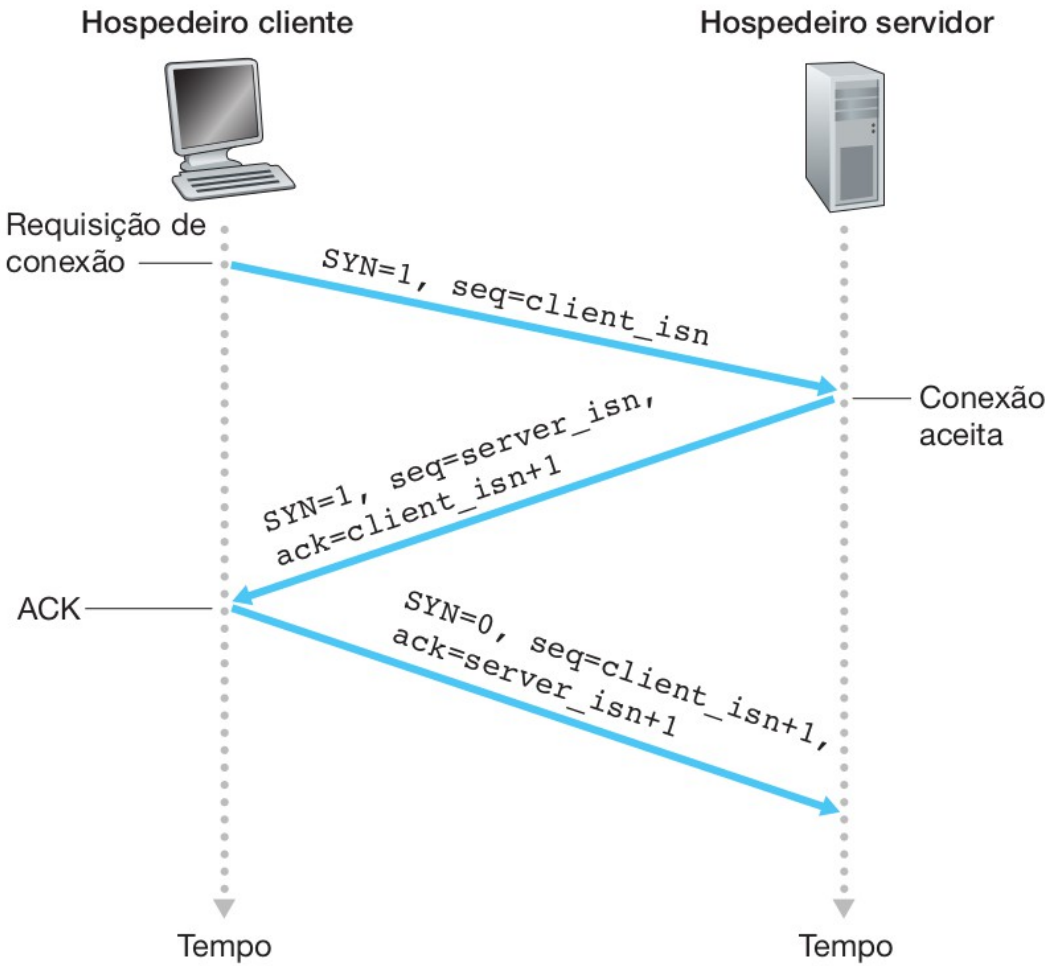


Gerenciamento de conexão

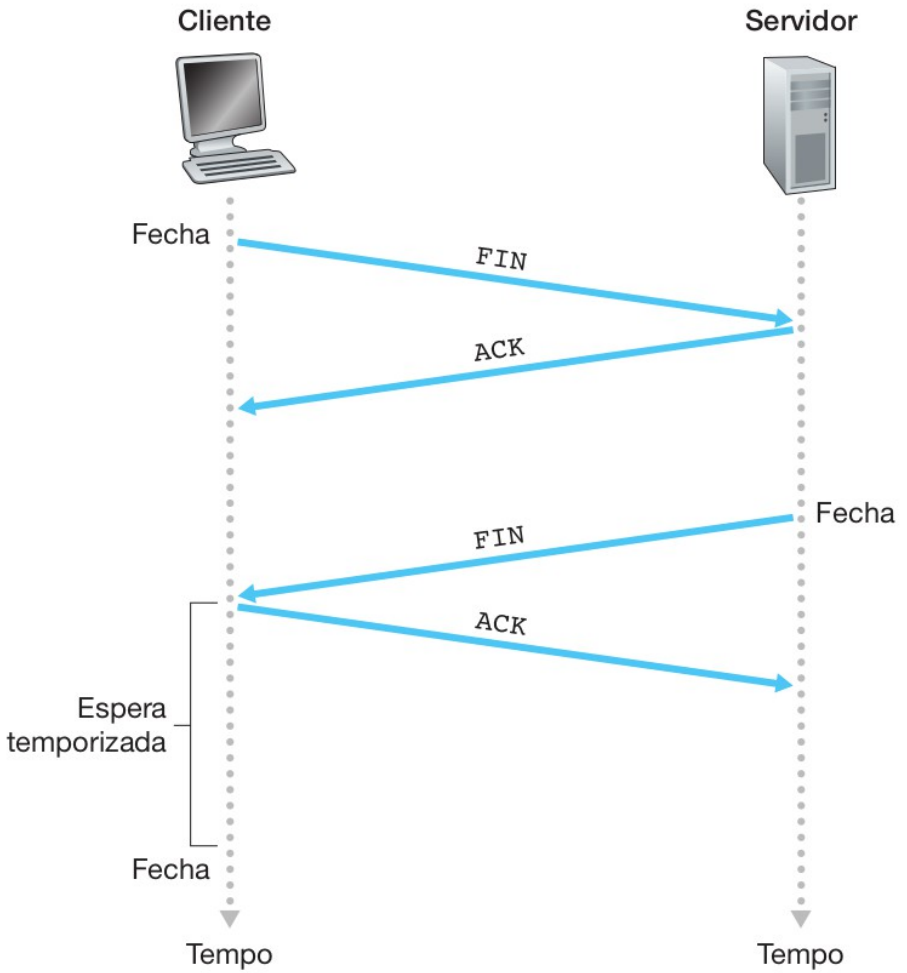
- Antes de trocar dados, transmissor e receptor TCP dialogam:
  - concordam em estabelecer uma conexão (cada um sabendo que o outro quer estabelecer a conexão)
  - concordam com os parâmetros da conexão.



Apresentação de 3 vias do TPC



Encerramento de uma conexão TCP



Redes de Computadores

Capítulo 3: Camada de transporte

Princípios de controle de congestionamento

Princípios de Controle de Congestionamento

Congestionamento:

- Informalmente: “muitas fontes enviando dados acima da capacidade da rede de tratá-los”

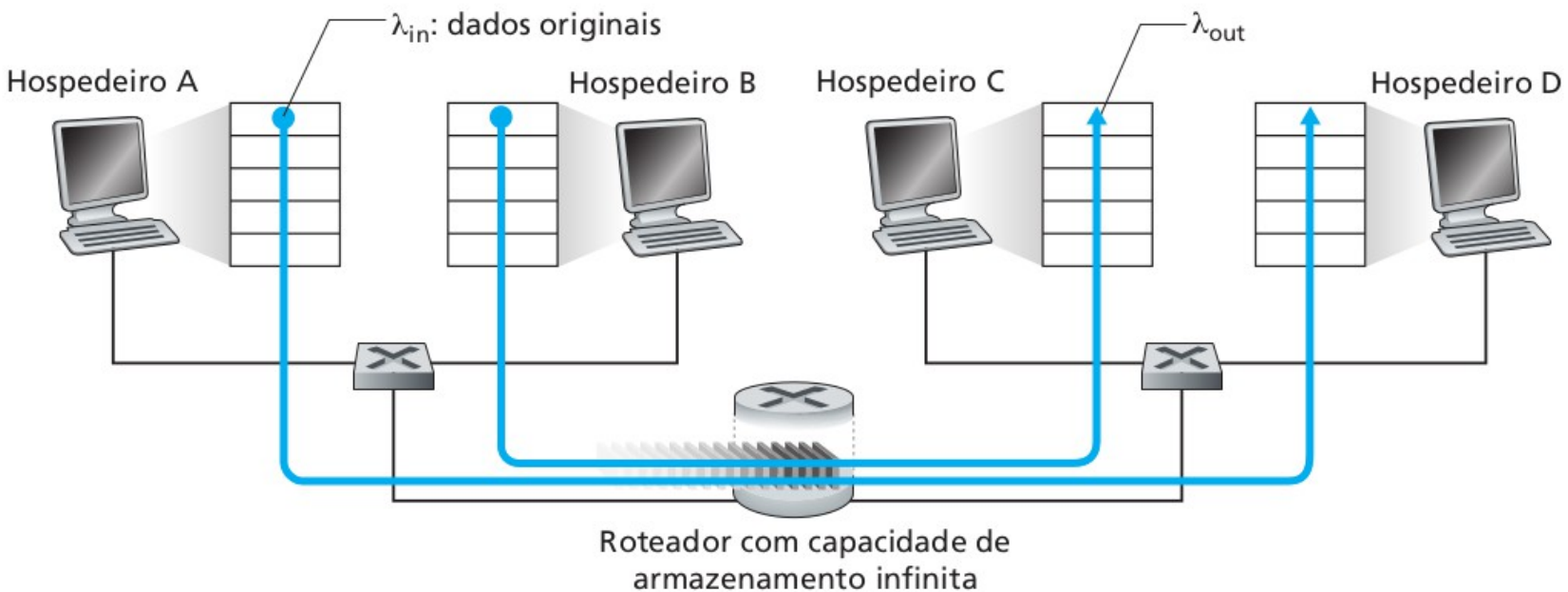
- Sintomas:
  - perda de pacotes (saturação de buffers nos roteadores)
  - longos atrasos (enfileiramento nos buffers dos roteadores)
  - um dos 10 problemas mais importantes em redes!

Princípios de Controle de Congestionamento

Controle de fluxo

Controle de congestionamento

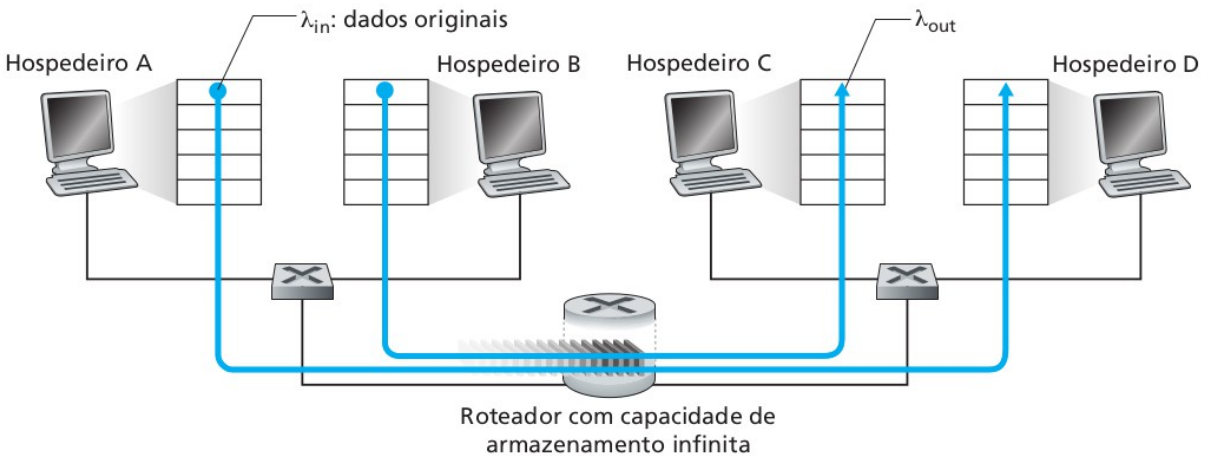
Causa/custo de congestionamento: cenário 1



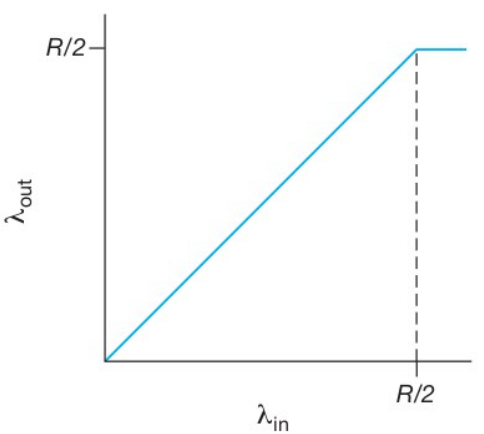
- Dois remetentes e dois receptores
- Um roteador com buffer infinito

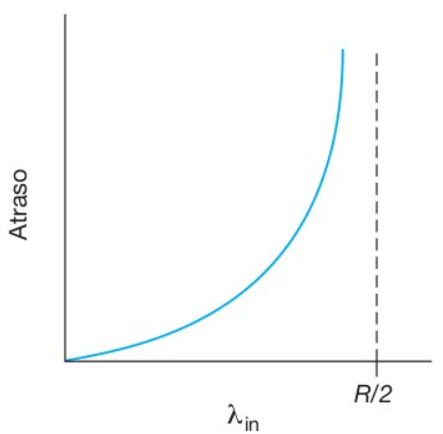
- Sem retransmissão
- Capacidade do link de saída: R

Causa/custo de congestionamento: cenário 1



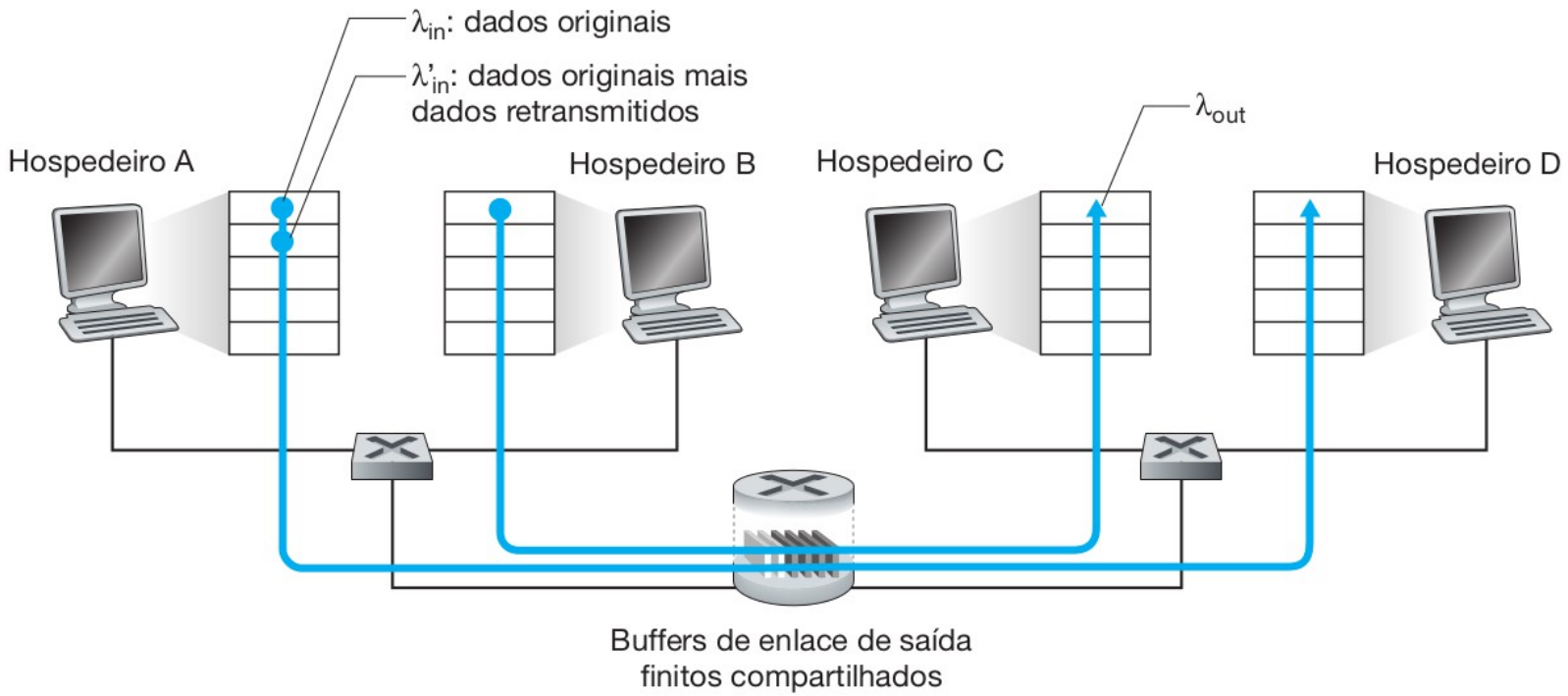
Vazão máxima por conexão: R/2





Grandes atrasos quando a taxa de chegada se aproxima da capacidade

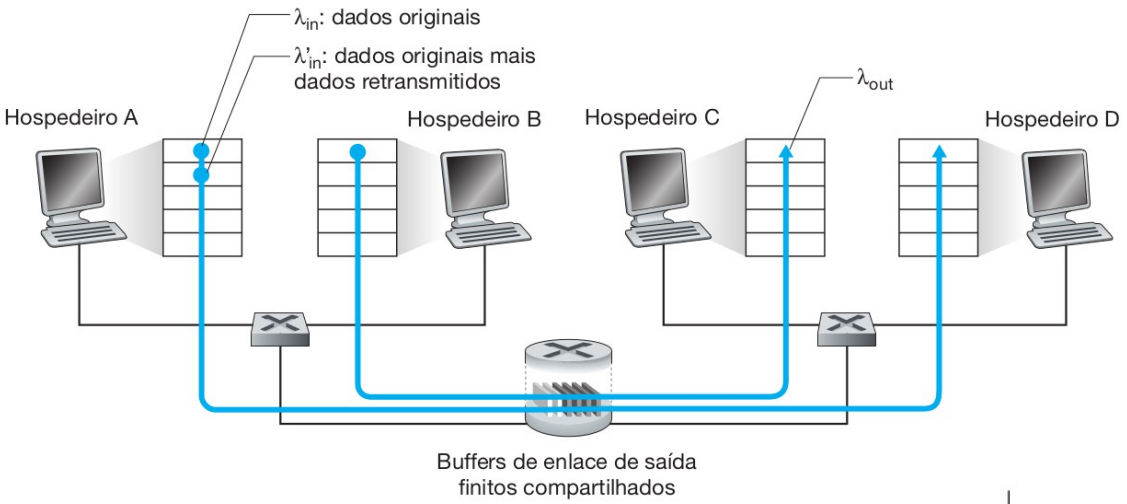
Causa/custo de congestionamento: cenário 2



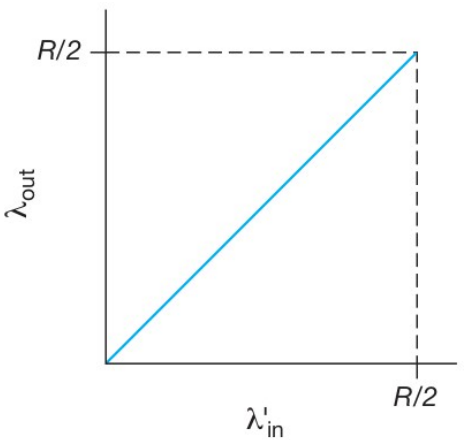
- Um roteador com buffers finitos
- Retransmissao pelo remetente de pacote perdido

- $\lambda_{in} = \lambda_{out}$
- λ'_{in} λ_{in}

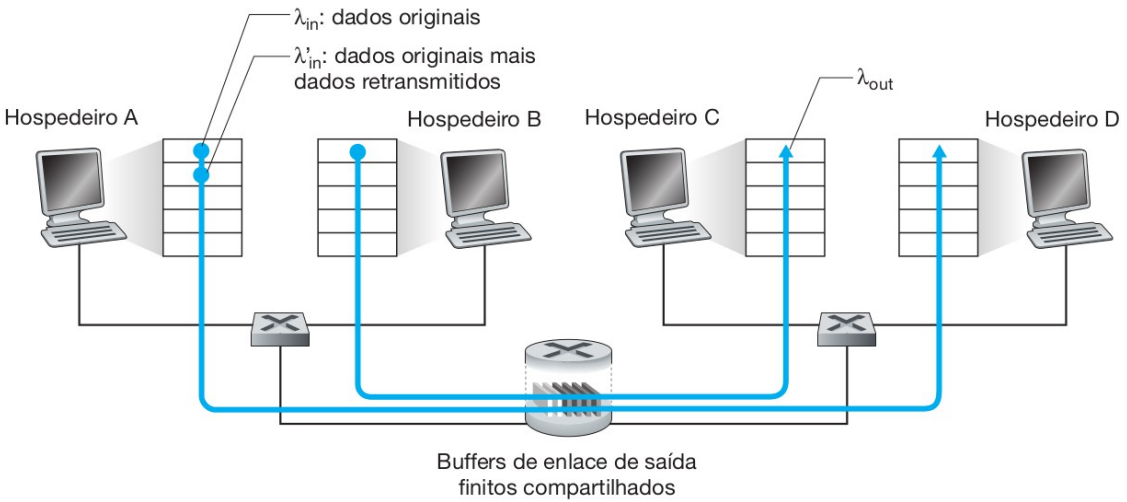
Causa/custo de congestionamento: cenário 2



- Idealização: conhecimento perfeito
  - transmissor envia apenas quando houver buffer disponível no roteador

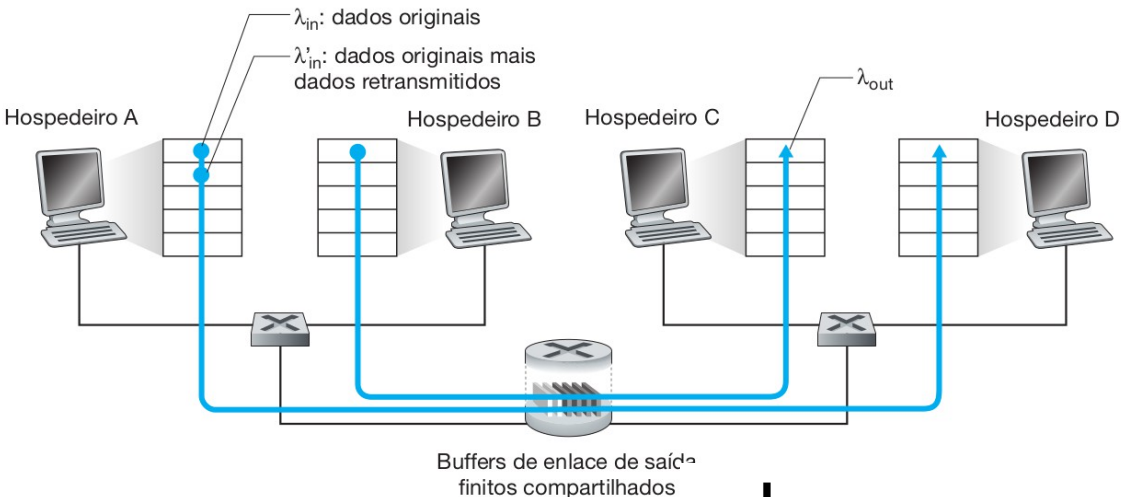


Causa/custo de congestionamento: cenário 2

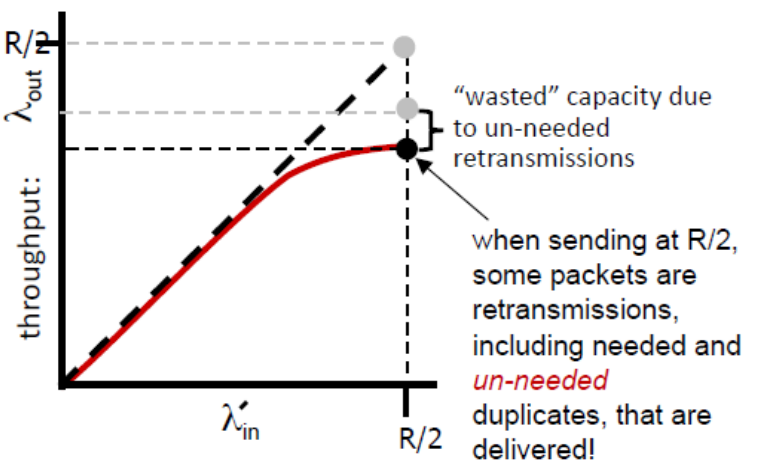


- Idealização: perda conhecida
  - pacotes podem ser perdidos, descartados no roteador devido a buffers cheios
  - transmissor apenas retransmite se o pacote sabidamente se perdeu.

Causa/custo de congestionamento: cenário 2

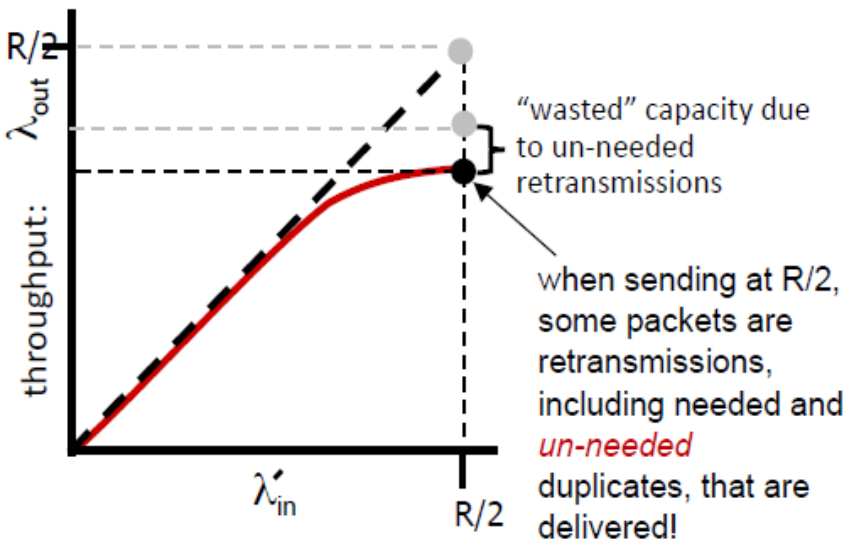


- Idealização: perda conhecida
  - pacotes podem ser perdidos, descartados no roteador devido a buffers cheios
  - transmissor apenas retransmite se o pacote sabidamente se perdeu.



Causa/custo de congestionamento: cenário 2

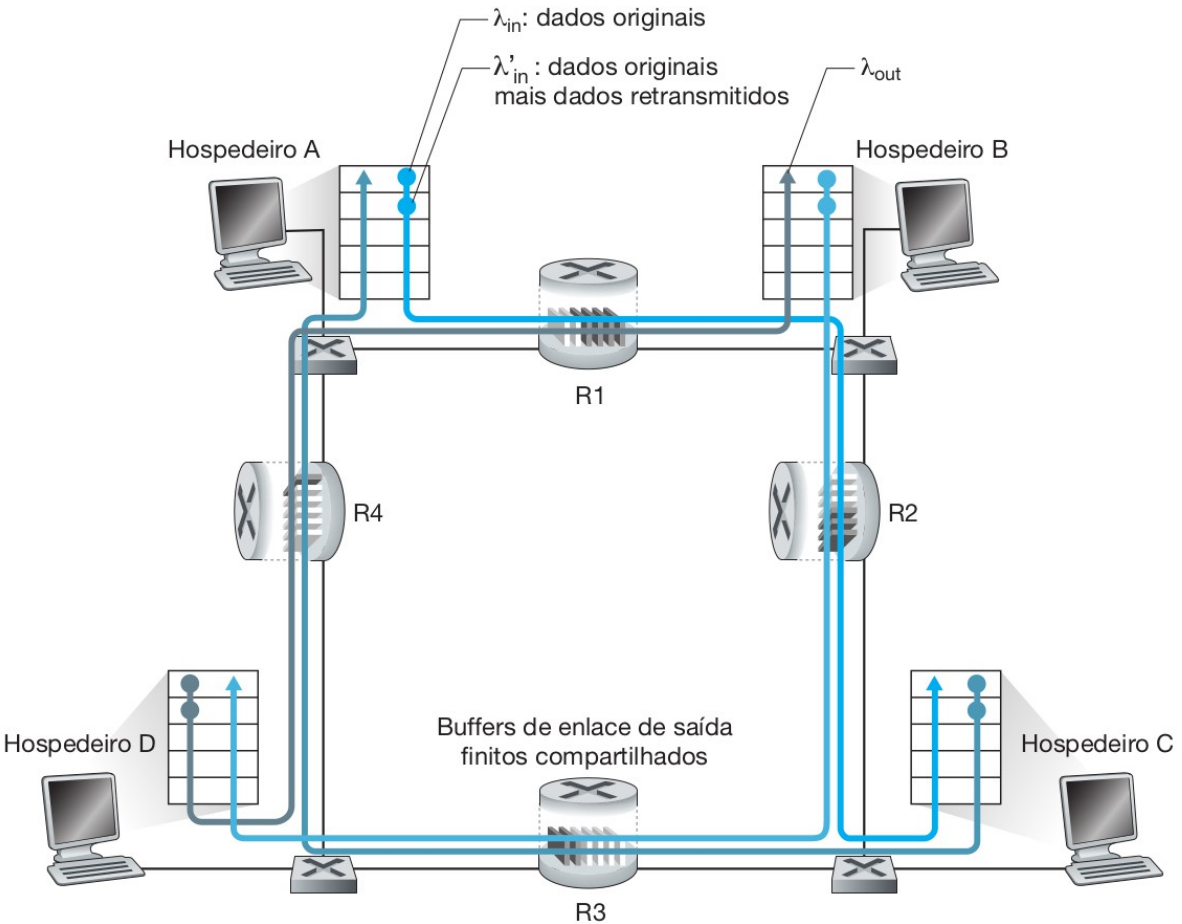
- Realidade: duplicatas
  - pacotes podem ser perdidos, descartados no roteador devido a buffers cheios
  - retransmissão prematura, envio de duas cópias, ambas entregues



- Custo do congestionamento:
  - mais trabalho (retransmissões) para uma dada "goodput"
  - Retransmissões desnecessárias: link transporta múltiplas cópias do pacote diminuindo a "goodput"

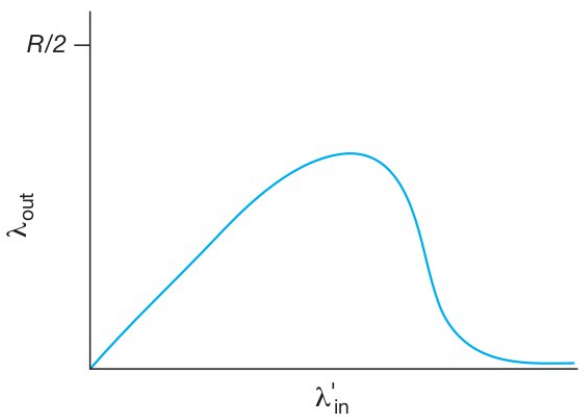
Causa/custo de congestionamento: cenário 3

- quatro remetentes
- caminhos com múltiplos enlaces e roteadores
- temporização/retransmissão

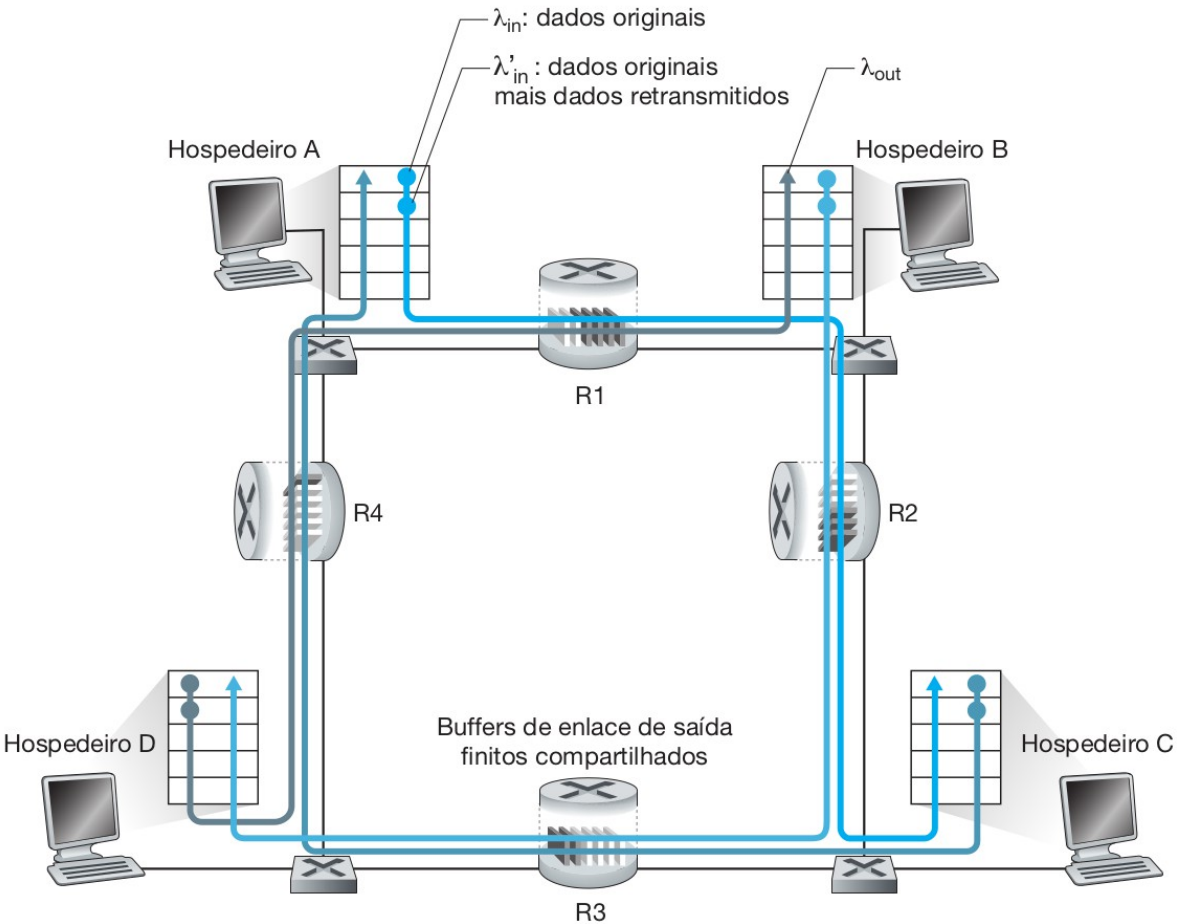


Causa/custo de congestionamento: cenário 3

- Desempenho?



Outro “custo” de congestionamento:
quando pacote é descartado, qualquer capacidade de transmissão já usada (antes do descarte) para esse pacote foi desperdiçada!



Redes de Computadores

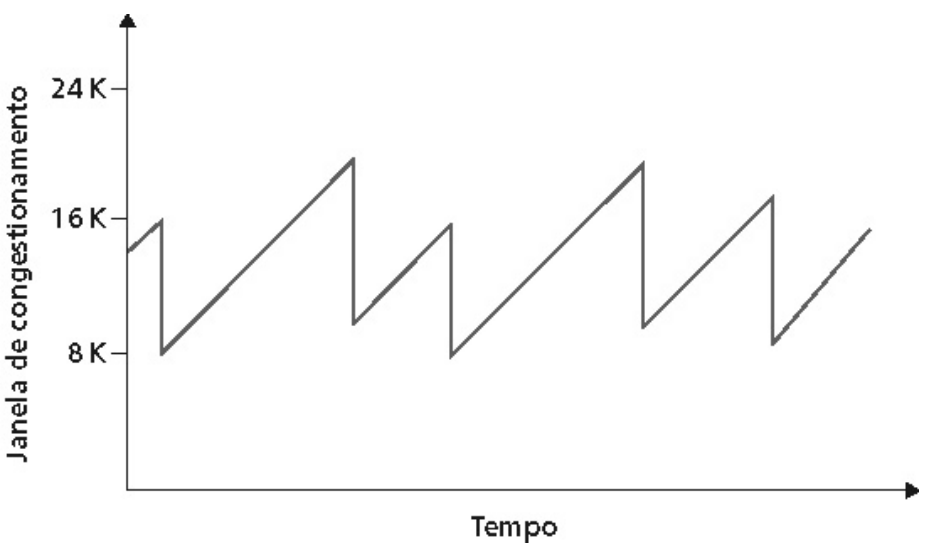
Capítulo 3: Camada de transporte

Controle de congestionamento do TCP

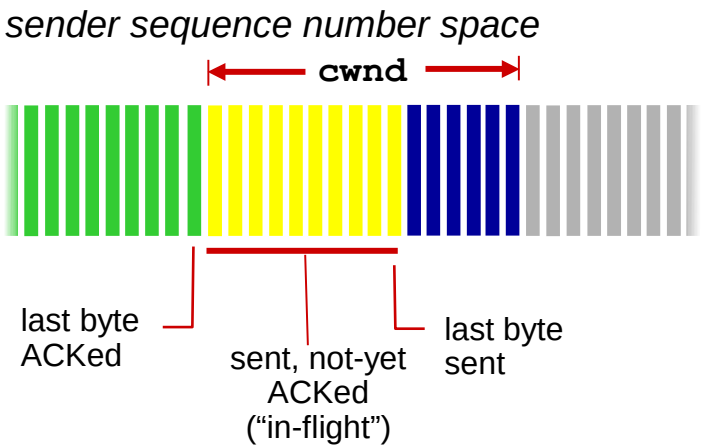
Controle de congestionamento do TCP

- Abordagem: aumentar a taxa de transmissão (tamanho da janela), testando a largura de banda utilizável, até que ocorra uma perda
  - aumento aditivo: incrementa cwnd de 1 MSS a cada RTT até detectar uma perda
  - diminuição multiplicativa: corta cwnd pela metade após evento de perda

Teste contínuo da largura de banda disponível



Controle de congestionamento do TCP



- Transmissor limita a transmissao:

  LastByteSent - LastByteAcked < cwnd

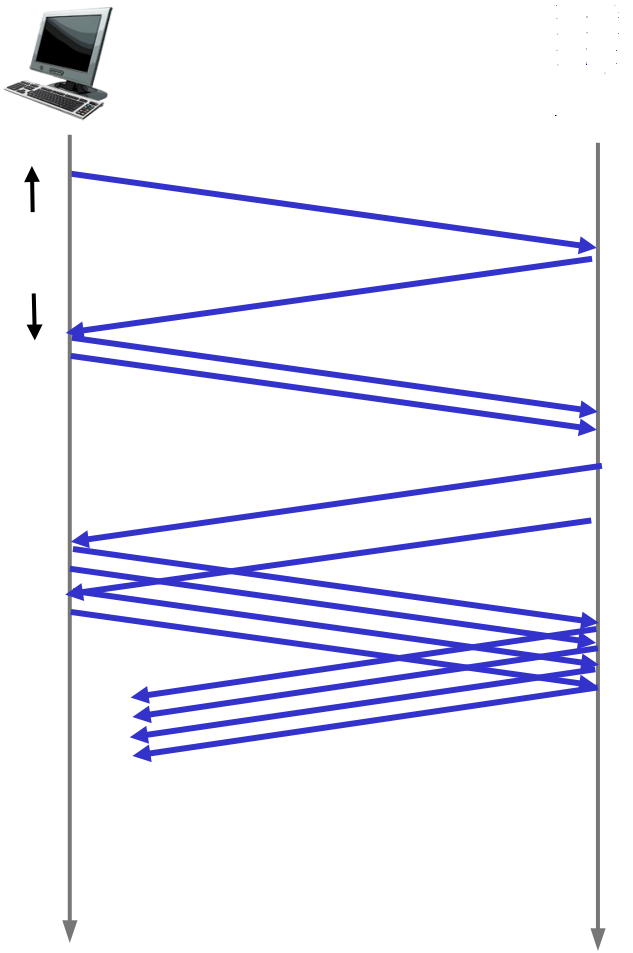
- **cwnd** é dinâmica, em função do congestionamento detectado na rede

- Taxa de transmissão do TCP:
  - aproximadamente : envia uma janela (cwnd), espera RTT para os ACKs, depois envia mais bytes

$$\text{taxa} = \frac{\text{cwnd}}{\text{RTT}} \quad \textit{bytes/seg}$$

Partida lenta do TCP

- no início da conexão, aumenta a taxa exponencialmente até o primeiro evento de perda:
  - inicialmente cwnd = 1 MSS
  - duplica cwnd a cada RTT
  - através do incremento da cwnd para cada ACK recebido
- resumo : taxa inicial é baixa mas cresce rapidamente de forma exponencial

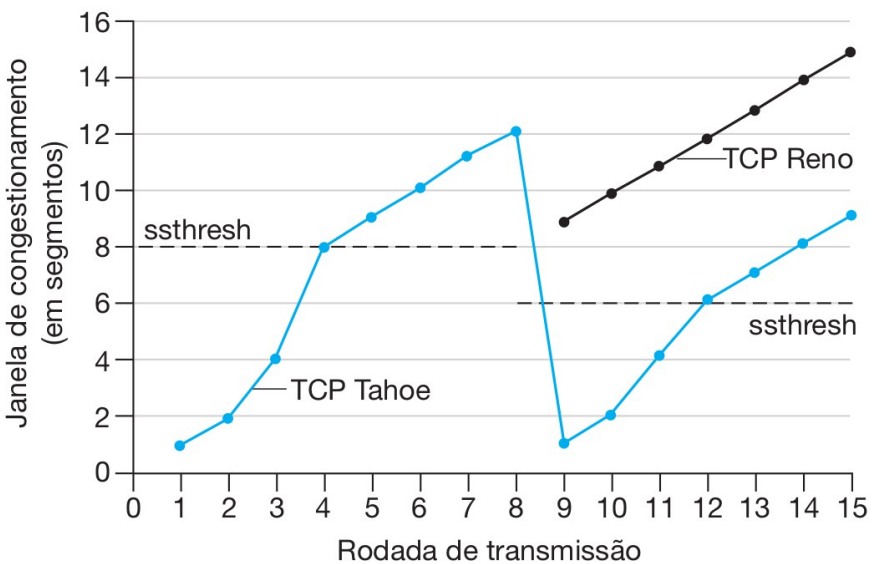


TCP: detectando, reagindo a perdas

- Perda indicada pelo estouro de temporizador:
  - cwnd é reduzida a 1 MSS;
  - janela cresce exponencialmente (como na partida lenta) até um limiar, depois cresce linearmente
- Perda indicada por ACKs duplicados: TCP RENO
  - ACKs duplicados indicam que a rede é capaz de entregar alguns segmentos
  - corta cwnd pela metade depois cresce linearmente
- O TCP Tahoe sempre reduz a cwnd para 1 (seja por estouro de temporizador que três ACKS duplicados)

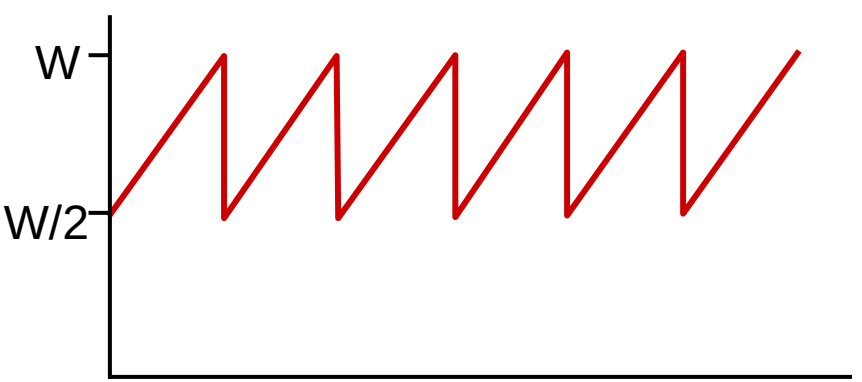
TCP: evolução da janela de congestionamento

- Pergunta: Quando o crescimento exponencial deve mudar para linear?
- Resposta: Quando cwnd atingir 1/2 do seu valor antes da detecção de perda.
- Implementação:
  - Limiar variável (ssthresh)
  - Com uma perda o limiar (ssthresh) é ajustado para 1/2 da cwnd imediatamente antes do evento de perda



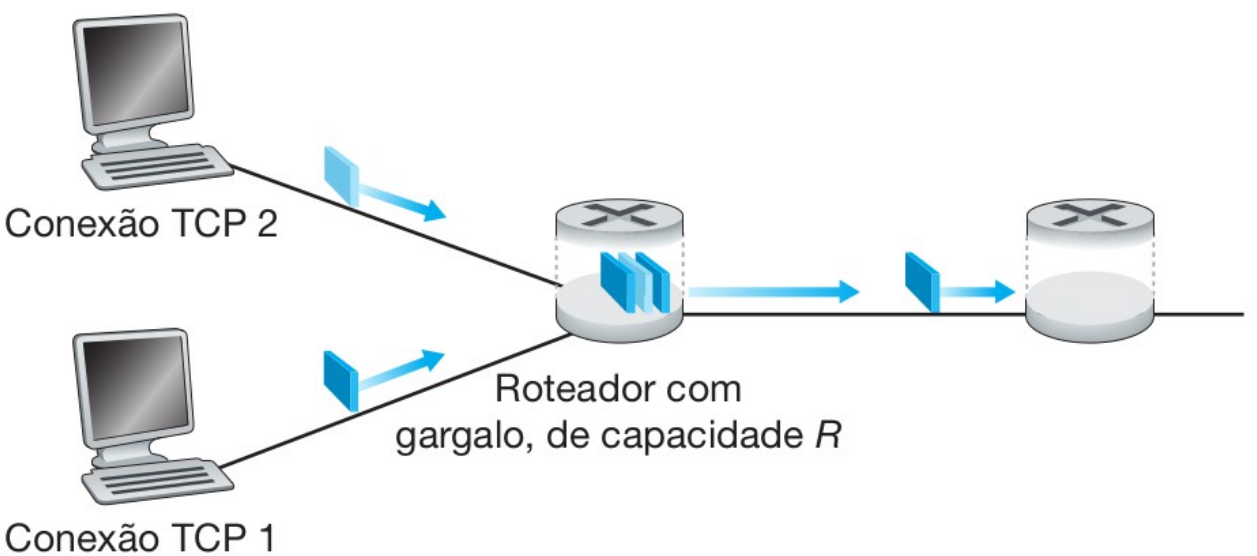
Vazão (throughput) do TCP

- Qual é a vazão média do TCP em função do tamanho da janela e do RTT
  - Ignore a partida lenta, assuma que sempre haja dados a serem transmitidos
- Seja W o tamanho da janela (medida em bytes) quando ocorre uma perda
  - Tamanho médio da janela é 3⁄4 W
  - Vazão média é de 3⁄4 W por RTT



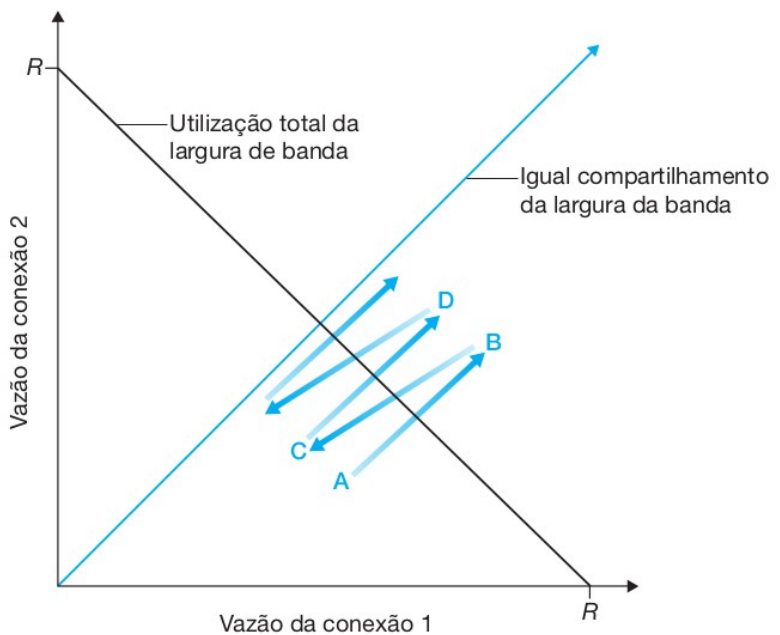
Equidade do TCP

- Objetivo de equidade: se K sessões TCP compartilham o mesmo enlace de gargalo com largura de banda R, cada uma deve obter uma taxa média de R/K



Por que o TCP é justo?

- Duas sessões competindo pela banda
  - Aumento aditivo dá gradiente de 1, enquanto vazão aumenta
  - Redução multiplicativa diminui vazão proporcionalmente



Equidade

Equidade e UDP

- aplicações multimídia frequentemente não usam TCP
  - não querem a taxa estrangulada pelo controle de congestionamento
- preferem usar o UDP:
  - injetam áudio/vídeo a taxas constantes, toleram perdas de pacotes

Equidade e conexões TCP em paralelo

- nada impede que as apls. Abram conexões paralelas entre 2 hosts
- os browsers Web fazem isto
- exemplo: canal com taxa R compartilhado por 9 conexões
  - novas aplicações pedem 1 TCP, obtém taxa de R/10
  - novas aplicações pedem 11 TCPs, obtém taxa R/2 !